1972

An Integrated Circuit BCH Cyclic Code Decoder

Rasik Desai

AN INTEGRATED CIRCUIT BCH CYCLIC CODE DECODER

BY

RASIK DESAI

A thesis submitted
in partial fulfillment of the requirements for the
degree Master of Science, Department of
Electrical Engineering, South Dakota
State University

1972

AN INTEGRATED CIRCUIT BCH CYCLIC CODE DECODER

This thesis is approved as a creditable and independent investigation by a candidate for the degree, Master of Science, and is acceptable as meeting the thesis requirements for this degree, but without implying that the conclusions reached by the candidate are necessarily the conclusions of the major department.

Thesis Adviser Date

Head, Electrical Engineering Department Date

ACKNOWLEDGMENT

The author wishes to thank and express his sincere appreciation to Dr. A. J. Kurtenbach for his assistance and constant encouragement in this study.

The author also wishes to thank the Center of Power System Study at South Dakota State University for partially supporting the research.

R. D.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY OF NOTATIONS

Symbol	Meaning
n	Total number of binary digits in a codeword
k	Number of information bits
r	Number of parity check bits
R	Information rate
C	Transmitted codeword
$\underline{G}$	Generator matrix
$\underline{I}_k$	k x k Identity matrix
a_i, b_i	Binary digits (0 or 1)
$\underline{H}$	Parity check matrix
$\underline{u}$	Transmitted code vector
$\underline{r}$	Received code vector
$\underline{e}$	Error vector
S	Syndrome
$w(v)$	Hamming Weight
$d(u, v)$	Hamming Distance
d_{min}	Minimum distance of a code
t	Number of errors to be corrected
l	Number of errors to be detected
$V(x)$	Code polynomial
v_i	Coefficients of code polynomial
$g(x)$	Generator polynomial
g_i	Coefficient of generator polynomial

Symbol	Meaning
$m(x)$	Message polynomial
m_i	Coefficients of message polynomial
$h(x)$	Parity check polynomial
h_i	Coefficient of parity check polynomial
$r(x)$	Remainder polynomial
r_i	Coefficient of remainder polynomial
$s(x)$	Syndrome polynomial
m	Degree of Primitive polynomial
a	Primitive element
$M_i(x)$	Minimum polynomial of a^i
GF	Galois Field
$\sigma(x)$	Errcr location polynomial
σ_i	Elementary Symmetric functions
S_i	Power sum symmetric functions
B_i	Error location numbers
$\triangle$	Determinant of a matrix
$p(x)$	Primitive polynomial
LCM	Least common multiple

CHAPTER I

INTRODUCTION

1.1. Problem Origin

The work done in this paper was motivated by a project that the United States Bureau of Reclamation has initiated to implement certain communication and control requirements.

The Colorado River storage project, under the Bureau of Reclamation, has a central power operations center at Montrose, Colorado, to serve as a computing and dispatch center for the surrounding power plants and substations. The requirement was to provide supervisory control facilities at the Montrose Center to control the encompassing substations and power plants. Specifically, the project was to furnish the dispatch center at Montrose with a stored program, programmable master station, which will have the capability of performing control, indication, alarm, and data transmission to and from the various power plants and substations.

This paper mainly deals with the last aspect of this project, namely, data transmission to and from other points.

The different data to be transmitted to and from the dispatch center include:

1. Voltage level, derived from power system potential transformers.
2. Current level, derived from power system current transformers.
3. Spillway gate position indication, derived from the rotation

of an intermediate gate hoist shaft which will drive a shaft to digital encoder, furnishing a digitized input to the supervisory control equipment.

4. Reservoir level, derived from a water-surface detector which will drive a shaft-to-digital encoder.

5. Governor gate limit position, derived from the rotation of the governor gate limit mechanism which will drive a shaft-to-digital encoder.

6. Outlet gate position indication, working as described above.

7. Tailwater level, derived from a float operated type gate which will drive a shaft-to-digital encoder.

8. Raise-lower command signals to the following.

   a. Power plant governing wicket gate limit.

   b. Power plant generator speed changer limit.

   c. Power plant generator voltage level.

9. Trip-close command signals to circuit breaker.

10. Load and frequency control system ON-OFF commands.

It is essential that the command signals received at the remotely controlled stations have the same form as when they were transmitted. In other words, it is essential that the message received at a receiving station should be error-free, or if it has errors, they should be corrected before it is fed to the devices concerned. The problem, thus originated out of the necessity of

having a digital data transmission system, that will be reliable, efficient, and at the same time, economically feasible.

1.2. Basic Coding Principle

Communication in a very broad sense, implies the transferring of information, in some form, from one point to another. The source of information may be natural or man-made. If the information to be transferred is in a numerical form, the system is said to be a digital data communication system.

In recent years, the demand for efficient and reliable digital data transmission systems has greatly increased, because of the widespread use of automatic data processors and the rising need for long range communication.

One of the serious problems in any high speed data transmission system is the occurrence of errors. The errors occur during the transmission of signals containing useful information through transmission channels. The transmission channel may be a telephone line, high frequency radio link, space communication link, or a magnetic tape unit, including writing and reading heads for storage systems.

Although it is not possible to prevent the channel from causing errors, we can reduce their undesirable effect with the use of coding. The basic idea is simple. We take a set of k message digits which we wish to transmit, annex to them r check digits and transmit the entire block of $n = k + r$ channel digits. Assuming that the channel noise

changes sufficiently few of these n transmitted channel digits, the r check digits may provide the receiver with sufficient information to enable it to detect and correct the channel errors.

Given any particular sequence of k message digits, the transmitter must have some rules for selecting the r check digits. This constitutes the "ENCODING PROBLEM." Any particular sequence of n digits which the encoder might transmit is called a codeword.

Although there are 2^n different binary sequences of length n, only 2^k of these sequences are codewords because the r check digits are completely determined by the k message digits. The set consisting of these 2^k codewords is called a code.

No matter which codeword is transmitted, any of the 2^n possible binary sequences of length n may be received if the channel is sufficiently noisy. Given the n received digits, the decoder must attempt to decide which of the 2^k possible codewords was transmitted.

Among the simplest examples of binary codes are the repetition codes. Here $k = 1$, r is arbitrary and $n = k + r = r + 1$. The code has only two codewords, a set of n zeroes or n ones. The value of each check digit is identical to the value of the message digit, 1 or 0.

The decoder might use the following rule. Count the number of zeroes and number of ones in the received word. Let p = number of zeroes; q = number of ones. Then if $p > q$, the codeword had all zeroes; $p < q$, the codeword had all ones; if $p = q$, do not decide. Thus, if $r = 5$, and if we want to transmit the digit 1, the codeword is 1 1 1 1 1 1.

If the received word is a corrupted version of the codeword, say 1 1 0 1 0 1, the decoder will count the number of zeroes ($p = 2$) and ones ($q = 4$) and since $p < q$, the codeword had all ones.

It is clear that this decoding rule will decode correctly in all cases when the channel noise changes less than half of the digits in any one block. If exactly half of the digits in any one block are changed, decoding failure will occur. If more than half of the digits are changed, a decoding error will result, i.e., the decoder will decode the received word into a wrong codeword.

In some applications a decoding error is tantamount to a disaster. For example, it may result in an incorrect command being received by a circuit breaker, an ICBM or a spaceship. A decoding failure on the other hand may result in the command's being ignored. This may represent only a minor nuisance, which can be overcome simply by repeating the command.

In such applications, however, one prefers a very incomplete decoding algorithm which intentionally refuses to decode any sufficiently ambiguous received word. One example of an incomplete decoding algorithm follows.

In the repetition code: let $n = 5 (r = 4, k = 1)$. The decoder will decode all received sequences containing 0 or 1 one into an all-zero codeword and all sequences containing 4 or 5 ones into all-one sequences. This decoding algorithm will fail to decode sequences having 2 or 3 ones. Although this incomplete decoding algorithm has a positive problem of decoding failure, it has considerably lower probability of decoding error.

The information digits are added according to the binary rules:

$0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 0$, $1 + 1 = 0$.

The binary sum of a number of binary digits is seen to be 0 or 1, accordingly as the number of ones among these digits is even or odd. It follows, then, that the total number of ones (including the check digit) in every codeword of a single parity-check code is even.

If the received word contains an even number of ones, the decoder may decode it without a change, but if the received word contains an odd number of ones, the decoder should not decode it. This incomplete decoding rule will decode correctly only if no channel errors occur in the transmitted block. A single channel error, or for that matter, any odd number of channel errors, will be detected as a decoding failure. Any combination of two channel errors, or any even number of channel errors (non-zero), will cause a decoding error.

These two examples, the repetition codes and the single-parity-check codes provide the extreme, relatively trivial, cases of binary codes. The repetition codes have enormous error-correction capability, but only one message digit per block, i.e., very low information rates. The single-parity check codes have very high information rates, but since they contain only one check digit per block, they are unable to do more than detect an odd number of channel errors.

In order to interpolate between these two extreme classes of codes to find codes which have moderate rates and moderate error correction capabilities, we shall consider the more general class of linear codes, of which repetition codes and single-parity-check codes are special cases.

1.3. A Typical Digital Data Communication System

A block diagram of a typical digital data communication system is shown in Figure 1.1. The first element of this system is the information source, which may be a person or a machine. The output of the source may be a continuous waveform or a sequence of discrete symbols (or letters).

The source encoder transforms the source output into a sequence m of binary symbols. This is the information sequence. The transformation should be done in such a way that: (1) the number of binary digits (bits) per unit of time required to represent the source output is small; and (2) the reconstruction or identification of the source output from the information sequence m is possible.

The channel is a medium over which signals containing useful information are transmitted. The channel is usually subject to various types of noise disturbances, natural or man made. For example, on a telphone line, the disturbance may come from thermal noise, lightning, impulse noise or cross talk from other lines. As pointed out earlier, the channel encoder, according to some rules,

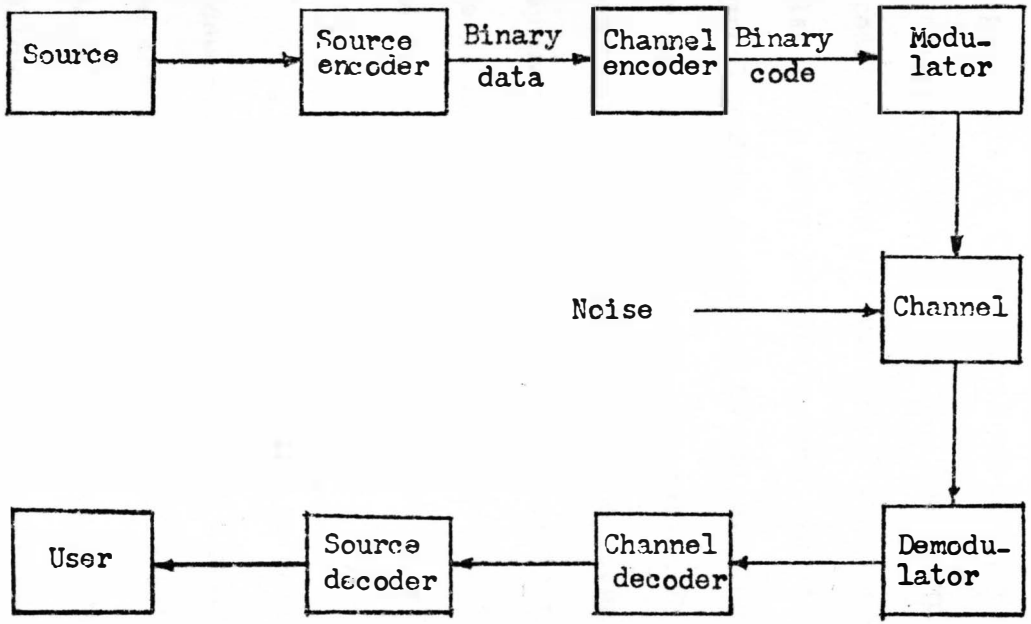


Figure 1.1

Data communication system

transforms the input information sequence m into some longer binary sequence c which is called the codeword.

The binary digits are not suitable for transmission over the physical channel. The function of the modulator is to encode each output digit of the channel encoder into one of two physical waveforms of duration T seconds. For example, a "1" may be encoded into a positive pulse of duration T seconds and "0" encoded into a negative pulse (or a blank). The output signal of the modulator enters the channel and is disturbed by noise. The demodulator makes a decision for each received signal of duration T, to determine whether a "1" or a "0" was transmitted. Thus, the output of the demodulator is a sequence r of binary digits. This is the received sequence. Due to the channel noise disturbance the received sequence might not match the codeword c. The places where they differ are called transmission errors.

The channel decoder should be designed such that its output codewords have the capability of combating the transmission errors. The channel decoder, based on the received sequence r, the rules of channel encoding, and the channel characteristics, does the following: (1) it attempts to correct the transmission errors in r, and produces an estimate c^* of the actual transmitted codeword, c; (2) it transforms c^* into an information sequence m^* which is an estimate of the transmitted information sequence m. The source decoder, based on the rules of source encoding, transforms m^* into an estimate s^* of the actual source output s and delivers it to

the user. If the channel is quiet, c,* m* and s* are reproductions of c, m and s respectively. If the channel is very noisy, s* might be quite different from the actual source output.

A major communication engineering problem is to design the channel encoder-decoder pair such that:

1. Binary data can be transmitted over the noisy channel as fast as possible; and

2. Reliable reproduction of the information sequence m can be obtained at the output of the channel decoder.

The design of the channel encoder-decoder pair is primarily based on the channel characteristics. For most practical purposes, a transmission channel called Binary Symmetric Channel (BSC) has been widely used. Here, we assume* that

$$q_0 > p_0$$

where

q_0 = probability of receiving the same symbol as the transmitted one

and

p_0 = probability of receiving the opposite symbol.

The transmission errors induced on the BSC are random errors. In general, random errors are one, in which all the transmitted symbols are affected independently by noise, i.e., they have equal probability of being changed.

* See Reference 3, Page 6.

In some cases, unlike BSC, several adjacent symbols are affected at the same time. These are burst errors and there are various codes to correct burst errors.

Sometimes the bursts come in bursts. A channel may be error-free for a long time and then very bad for a short while, but long enough to make error correction difficult. For such a channel, only limited improvement can be attained with error correction alone; and some combination of error correction with error detection and request for repeat is required.

In this paper, only random errors are taken into account and it is assumed that a request for a repetition of the message is not possible.

In a nutshell, the research work done in the paper to follow centers around designing an electronic device that would perform the following functions.

1. Process a message as read from the Master Station to detect presence of errors, if any.

2. If an erroneous message is detected, try to correct it. The device is generally known as BCH decoder named after Bose-Chaudhuri-Hoquenghem, who invented the BCH cyclic codes for this purpose.

CHAPTER II

TYPES OF CODES

In a digital data communication system, various codes with varying efficiency and transmission rates have been devised. In fact, it is possible to have an infinite number of error-correcting or error-detecting codes. This is because of the fact that to any particular message block can be attached a set of parity check digits (in a systematic way) in almost infinite ways, each one resulting in a unique type of code. However, for classification purposes all codes have been principally divided into two classes: Block codes and Convolutional codes. It should be pointed out here that in the material to follow, all plus signs (+) indicate the addition of binary symbols in Galois Field GF(2). For details see Appendix A.

2.1. Block Codes and Convolutional Codes

In block codes,[1] the block of n code digits generated by the encoder in any particular time unit depends only on the block of k input message digits within that time unit.

In convolutional codes,[1] the block of n code digits generated by the encoder in any particular time unit depends not only on the block of k message digits within that time unit, but also on the blocks of message digits within a previous span of (n - 1) time units (N > 1).

This paper addresses block codes in general and linear block codes in particular, a detailed treatment of which is given below.

2.2 Definitions - Coding Theory Terms

Block Code: When a sequence of binary information digits is divided into blocks of length k, we have 2^k possible message blocks. Corresponding to these 2^k possible message digits there will be 2^k possible codewords of length n ($n > k$), at the output of the encoder. This set of 2^k codewords forms a block code.

Linear block code: A set of 2^k n-tuples (or codewords of n digits) is called a linear block code, if it is a subspace of the vector space V_n of all n-tuples. As an example, consider an encoder that segments message blocks of two digits into encoded codewords of three digits.

Message	Codeword
0 0	0 0 1
0 1	0 1 0
1 0	1 0 0
1 1	1 1 1

Here $k = 2$. There are $2^k = 4$ possible messages, and hence 4 possible codewords of length 3 digits. Each codeword is distinct. Also, the set of codewords forms a subspace of the vector space of all 3-tuples. Therefore, they form a linear code.

Generator Matrix

A linear block code of 2^k code vectors can also be described by a set of k binary independent code vectors arranged as the rows of a k x n matrix. Alternately, we can describe this matrix as

$$\underline{G} = \left[\underline{I}_k \; \underline{P}\right]$$

where

$\underline{I}_k$ = k x k identity matrix

$\underline{P}$ = k x (n - k) matrix

i.e.,

$$\underline{G} = \begin{bmatrix} 1\,0\,0\,\ldots\,0 & p_{11} & p_{12} & \cdots & p_{1,n-k} \\ 0\,1\,0\,\ldots\,0 & p_{21} & p_{22} & \cdots & p_{2,n-k} \\ \vdots & & & & \\ 0\,0\,0\,\ldots\,1 & p_{k1} & p_{k2} & \cdots & p_{k,n-k} \end{bmatrix} \qquad (2.1)$$

where p_{ij} = 0 or 1 are determined according to the type of parity checks used. This is called the generator matrix of a given linear block code. The rows of $\underline{G}$ generate a linear code and hence $\underline{G}$ completely specifies a linear code.

Example 2.1: Consider an encoder which segments the information into message blocks of 3 digits and transforms each block into a code vector of 6 digits as follows.

Message	Codeword
$a_1\ a_2\ a_3$	$a_1\ a_2\ a_3\ a_4\ a_5\ a_6$
0 0 0	0 0 0 0 0 0
0 0 1	0 0 1 1 0 1
0 1 0	0 1 0 0 1 1

Message	Codeword
0 1 1	0 1 1 1 1 0
1 0 0	1 0 0 1 1 0
1 0 1	1 0 1 0 1 1
1 1 0	1 1 0 1 0 1
1 1 1	1 1 1 0 0 0

where

$$a_4 = a_1 + a_3$$
$$a_5 = a_1 + a_2$$
$$a_6 = a_2 + a_3$$

From the above, it is possible to find a set of k linearly independent vectors, such that the linear combinations of the k vectors give all the 2^k code vectors. Since these linearly independent vectors form a generator matrix of a code, the generator matrix in this case is

$$\underline{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Parity Check Matrix

For each k x n matrix $\underline{G}$, there exists a (k - n) x n matrix $\underline{H}$, such that the row space of $\underline{G}$ is orthogonal to $\underline{H}$; i.e., the inner product of a vector in the row space of $\underline{G}$ and a row of $\underline{H}$ is zero.

This matrix $\underline{H}$ is called the parity check matrix of the linear block code. It can be described as

$$\underline{H} = \underline{P}^T \; \underline{I}_{n-k}$$

where P^T is the transpose of matrix P

$$\text{i.e., } \underline{H} = \begin{bmatrix} p_{11} & p_{21} & \cdots & p_{k1} & 1\,0\,0\,\cdots\,0 \\ p_{12} & p_{22} & \cdots & p_{k2} & 0\,1\,0\,\cdots\,0 \\ \vdots & & & & \\ p_{1,n-k} & p_{2,n-k} & \cdots & p_{k,n-k} & 0\,0\,0\,\cdots\,1 \end{bmatrix} \qquad (2.2)$$

Systematic Code

A systematic code is one in which each codeword has the first k digits as the message bits and the last (n - k) digits as the parity check digits. Any code can be put into a systematic form; for details see the text by Shu Lin.[1]

2.3. Syndrome and Its Application

Consider a linear (n, k) code with generator matrix $\underline{G}$ and parity check matrix $\underline{H}$. Let $\underline{u}$ be a code vector transmitted over a noisy channel. At the receiving end, we might have a corrupted vector $\underline{r}$, which is a vector sum of the original code vector $\underline{u}$ and an error vector $\underline{e}$, i.e.

$$\underline{r} = \underline{u} + \underline{e}$$

The receiver does not know $\underline{u}$ and $\underline{e}$. The purpose of the decoder is to recover $\underline{u}$ from $\underline{r}$. The syndrome of a received vector is given by the (n - k) component vector

$$S = \underline{r}\,\underline{H}^T$$

where $\underline{H}^T$ is the transpose of the parity check matrix $\underline{H}$. The syndrome is zero if $\underline{r}$ is a code vector and is not zero if $\underline{r}$ is corrupted and is not a code vector. The syndrome is used to detect and correct channel errors, as will be clear from the following example.

Example 2.2: The generator matrix $\underline{G}$ of a (6, 3) code is given by

$$\underline{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} = \underline{I}_k \ \underline{P}$$

The parity check matrix is

$$\underline{H} = \underline{P}^T \ \underline{I}_{n-k} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Now, if the message is m = 1 1 1, then from Example 2.1, the corresponding code vector is 1 1 1 0 0 0 and the syndrome is

$$S = \underline{r} \ \underline{H}^T$$

$$= (1\ 1\ 1\ 0\ 0\ 0) \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= (1\ 1\ 0 + 0\ 1\ 1 + 1\ 0\ 1)$$

$$S = 0\ 0\ 0$$

This indicates that the received vector has no error and is a code vector of the (6, 3) code.

On the other hand, suppose the received vector is (1 1 1 1 0 0), then

$$S = (1\ 1\ 1\ 1\ 0\ 0)\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 1\ 0\ 0$$

This indicates that there is an error in the received codeword. There are several ways to correct this error, one of which is to use a standard array of the code under question. Shu Lin[1] and Peterson[3] have given a simplified approach to standard arrays.

2.4. Weight and Distance of a Codeword

The Hamming weight of an n-tuple is defined as the number of non-zero components in it. For example, if a codeword is 1 1 1 0 0 0, the Hamming weight w(v) = 3. The Hamming distance between two code vectors $\underline{u}$ and $\underline{v}$ is defined as the number of components in which they differ. Thus if

$$\underline{u} = 1\ 1\ 1\ 0\ 0\ 0$$

$$\underline{v} = 1\ 0\ 1\ 0\ 1\ 0$$

Then the Hamming distance is given by

$$d(\underline{u},\underline{v}) = 2$$

Note that the Hamming distance between $\underline{u}$ and $\underline{v}$ is just equal to the weight of their vector sum, $\underline{u} + \underline{v}$, i.e.

$$d(\underline{u},\underline{v}) = w(\underline{u} + \underline{v})$$

The minimum distance between any pair of codewords of a linear code is the minimum distance of the code, d_{min}. Alternately, the

minimum distance of a linear code is equal to the minimum weight of its non-zero code vectors. The concept of minimum distance of a code is very important since it determines the error-correcting capability of a code.

2.5 Error-Correcting and Error Detecting Capability of a Binary Code

If a code with minimum distance, d_{min}, such that $2t + 2 \geqslant d_{min} \geqslant 2t + 1$, is used for random error correction, the decoder will correct all codewords with t or fewer errors, which may occur during the transition.

In general, the error correcting capability of a linear code with minimum distance d_{min} is given by

$$t = (d_{min} - 1)/2$$

where $(d_{min} - 1)/2$ denotes the largest integer no greater than $(d_{min} - 1)/2$. A code of error correcting capability 't' is generally called a t-error-correcting code. The error correcting capability defined above is with respect to random errors.

Random-Error Detection Capability

If a code with a minimum distance d_{min} is used for straight error detection, the decoder can detect all error-patterns of $(d_{min} - 1)$ or fewer errors. If a code is used for simultaneous correction of all combinations of t or fewer errors and detection of all combinations of $l \geqslant t$ errors, then the code is required to have a minimum distance $(t + l + 1)$. Thus, for a given n and k,

we would like to design an (n, k) code with minimum distance as large as possible (for random error correction). There is no systematic approach known so far to accomplish this.

2.6 Cyclic Codes

Cyclic codes are a subclass of linear block codes. The advantages of cyclic codes are:

1. Easy encoding and syndrome calculations using shift registers with feedback.

2. It is possible to find various simple and efficient decoding methods because of their inherent algebraic structure.

The theorems and definitions that follow contribute to a better understanding of the properties of cyclic codes. No attempt has been made to give proofs of the theorem. Those interested in details may refer to various books on this topic, especially those by Shu Lin[1] and Peterson.[3]

Theorem 1[1]

An (n, k) linear code is called a cyclic code if by shifting a code vector one place to the right, the resulting block is also a code vector. Thus, for a (7, 4) cyclic code, if

$\underline{v}$ = 0011010 is a code vector

then v^1 = 0001101 obtained by shifting one place to the right is also a code vector.

Because of their algebraic structure, the components of a code vector can be treated as coefficients of a polynomial. If

$\underline{V} = (v_0, v_1, v_2 \ldots, v_{n-1})$ is a codeword then, the polynomial is given by

$$V(x) = v_0 + v_1 x + v_2 x^2 + \ldots + v_{n-1} x^{n-1}$$

The terms, code vector and code polynomial, are used interchangeably in this paper.

Theorem 2[1]

In an (n, k) cyclic code, there exists one and only one polynomial g(x) of degree (n - k), given by

$$g(x) = 1 + g_1 x + g_2 x^2 + \ldots + g_{n-k-1} x^{n-k-1} + x^{n-k}$$

Every code polynomial V(x) is a multiple of g(x), and every polynomial of degree (n - 1) or less which is a multiple of g(x) may be a code polynomial.

It follows from Theorem 2, that every code polynomial V(x) in an (n, k) cyclic code can be expressed in the following form.

$$V(x) = m(x)\, g(x)$$

$$= (m_0 + m_1 x + m_2 x^2 + \ldots + m_{k-1} x^{k-1})\, g(x)$$

If the coefficient of m(x) are the k information digits to be encoded, then V(x) would be the corresponding code polynomial. Then the encoding of message m(x) is equivalent to multiplying the message m(x) by g(x).

The polynomial g(x) is the generator polynomial of the cyclic code and has degree (n - k) = number of parity check digits. An (n, k) cyclic code can be completely specified by the generator polynomial g(x).

Theorem 3[1]

The generator polynomial g(x) of an (n, k) cyclic code is a factor of $(x^n + 1)$, i.e.,

$$(x^n + 1) = g(x)\, h(x)$$

Theorem 4[1]

If g(x) is a polynomial of degree (n - k) and is a factor of $(x^n + 1)$, then g(x) generates a cyclic code.

Example 2.3: Consider a (7, 4) linear block code. If it is a cyclic code, it should be possible to factorize $(x^7 + 1)$.

$$(x^7 + 1) = (1 + x + x^3)\,(1 + x + x^2 + x^4)$$

To select g(x), we see the factor with degree (n - k). $(1 + x + x^3)$ has degree (n - k) and is a factor of $(x^7 + 1)$. Therefore, $g(x) = 1 + x + x^3$ is the generator polynomial of a (7, 4) cyclic code.

For this (7, 4) cyclic code there would be $2^k = 2^4 = 16$ possible message blocks, and hence 2^k possible code polynomials. Let the message block be

$$m = 0001.$$

Then $m(x) = 0.x^0 = 0.x^1 + 0.x^2 + 0.x^3 + 1.x^3 = x^3$

To find the encoded codeword in systematic form, multiply m(x) by x^{n-k} and divide the product by g(x).

$$x^3 m(x) = x^3 \cdot x^3 = x^6$$

$$x^3 + x + 1 \;\overline{)\; x^6 \;}$$

$$x^3 + x + 1 \;\overline{)\; x^6 \;\big|\; x^3 + x + 1}$$

$$\begin{array}{r|l} x^3 + x + 1 & x^6 \\ & \underline{x^6 + x^4 + x^3} \\ & x^4 + x^3 \\ & \underline{x^4 + x^2 + x} \\ & x^3 + x^2 + x \\ & \underline{x^3 + x + 1} \\ & x^2 + 1 \end{array}$$

Remainder $r(x) = x^2 + 1$.

The code polynomial in systematic form is then

$$V(x) = r(x) + x^{n-k} m(x)$$

$$V(x) = x^2 + 1 + x^6$$

i.e., $\underline{V} = 1\ 0\ 1\ 0\ 0\ 0\ 1$

Syndrome Calculation

As said earlier, the function of the decoder is to recover the transmitted codeword from the knowledge of received code vector.

The decoder first tests whether or not the received vector is a code vector by calculating the syndrome. If the syndrome is zero, the received vector is a code vector. To calculate the syndrome, the received vector r(x) is divided by g(x) and the remainder gives the syndrome, i.e., $r(x) = p(x)\, g(x) + s(x)$. For a received vector to be a code vector, $s(x) = 0$.

The structure and properties of cyclic codes were emphasized in this chapter. In the next chapter we will make use of some of these properties of cyclic codes to define BCH cyclic codes and elaborate on their properties.

CHAPTER III

BCH CYCLIC CODES

In 1959 Hocquenghem, A. and in 1960 Bose and Chaudhuri, working independently, discovered a cyclic code that is by far the most extensive and powerful code for random error correction. The code is named after these three men and called the BCH cyclic code.

The code was first developed for binary digits and for correcting two random errors. It has since been generalized to combat any t random errors in a channel.

This chapter is primarily devoted to describing the properties of binary BCH cyclic codes and their decoding algorithms.

3.1. Definition of BCH Code

For any positive integers m and t ($t < 2^m - 1$) there exists a BCH code such that

n = block length = $2^m - 1$

$(n - k)$ = number of check digits $\leq mt$

d_{min} = minimum distance $\geq 2t + 1$.

Since d_{min} gives the error correcting capability of a BCH code, this code can correct any combination of t or fewer random errors. For example:

let $m = 3,\ t = 1,$

then, $n = 2^m - 1 = 2^3 - 1 = 7$

$(n-k) \leq mt = 3$

$k = 7 - 3 = 4$

$d \geq 2t + 1 = 3$

Thus, a (7, 4) BCH cyclic code can correct a single error (t = 1).

The example above helps clarify the BCH code structure. It should, however, be pointed out that determination of the parity check digits (n - k) is not very straightforward. This will be explained in Section 3.2 that follows.

3.2. Generator Polynomial

As discussed in Appendix A, a primitive element of $GF(2^m)$ is one whose powers generate all the non-zero elements of $GF(2^m)$. It is shown that a is a primitive element of $GF(2^m)$. It can be shown that a^2, a^4 are also primitive elements of $GF(2^m)$. This is done with the irreducible polynomial $x^4 + x + 1 = 0$.

Let a be a primitive element* of $GF(2^m)$ and let $m_i(x)$ be the minimum polynomial of a^i (for i = 1, 3, 5, ..., 2t - 1). To find $m_i(x)$, use the method described in Appendix A. The generator polynomial of a t-error correcting BCH cyclic code is given by $g(x) = LCM (m_1(x), m_3(x), ..., m_{2t-1}(x))$. Since the degree of each $m_i(x)$ is m or less, the degree of $g(x) \leq mt$, i.e., $(n - k) \leq$ degree of g(x). For small t, (n - k) = mt.

Standard tables are available which give the parameters n and k for different t's. Tables are also available for the generator polynomials for different values of t. These are shown in Appendix B.

* Primitive element and other GF algebra is explained in Appendix A.

As t increases, the number of information bits decreases, or alternately, the transmission rate goes down (R = k/n) with an increase in t, for a fixed block length. This is apparent from the set of curves shown in Figure 3.1.

The following example illustrates how to compute the generator polynomial of a BCH code. Let a be a primitive element of $GF(2^4)$ (m = 4); let $m_1(x)$, $m_3(x)$ and $m_5(x)$ up to $m_{2t-1}(x)$ be the minimum polynomials of a, a^3 and a^5 respectively, for t = 3. Then,

$$m_1(x) = 1 + x + x^4 \text{ (refer to Appendix A).}$$

Similarly,

$$m_3(x) = 1 + x + x^2 + x^3 + x^4$$

$$m_5(x) = 1 + x + x^2.$$

The generator polynomial g(x) of this 3-error correcting code is given by

$$g(x) = \text{LCM } (m_1(x), m_3(x), m_5(x))$$

$$g(x) = \text{LCM } ((1 + x + x^4), (1 + x + x^2 + x^3 + x^4), (1 + x + x^2))$$

Since $m_1(x)$, $m_3(x)$ and $m_5(x)$ are irreducible,

$$g(x) = (1 + x + x^4)\ (1 + x + x^2 + x^3 + x^4)\ (1 + x + x^2)$$

Simplifying,

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1.$$

The degree of g(x) gives (n - k) and hence the number of parity check digits (n - k) = 10. Therefore, n = 15, k = 5 and this 3-error correcting code is a (15, 5) BCH cyclic code.

As a check, we find that $(n - k) = 10 \leq mt = 12$ and

g = 1 1 1 0 1 1 0 0 1 0 1 0 0 0 0

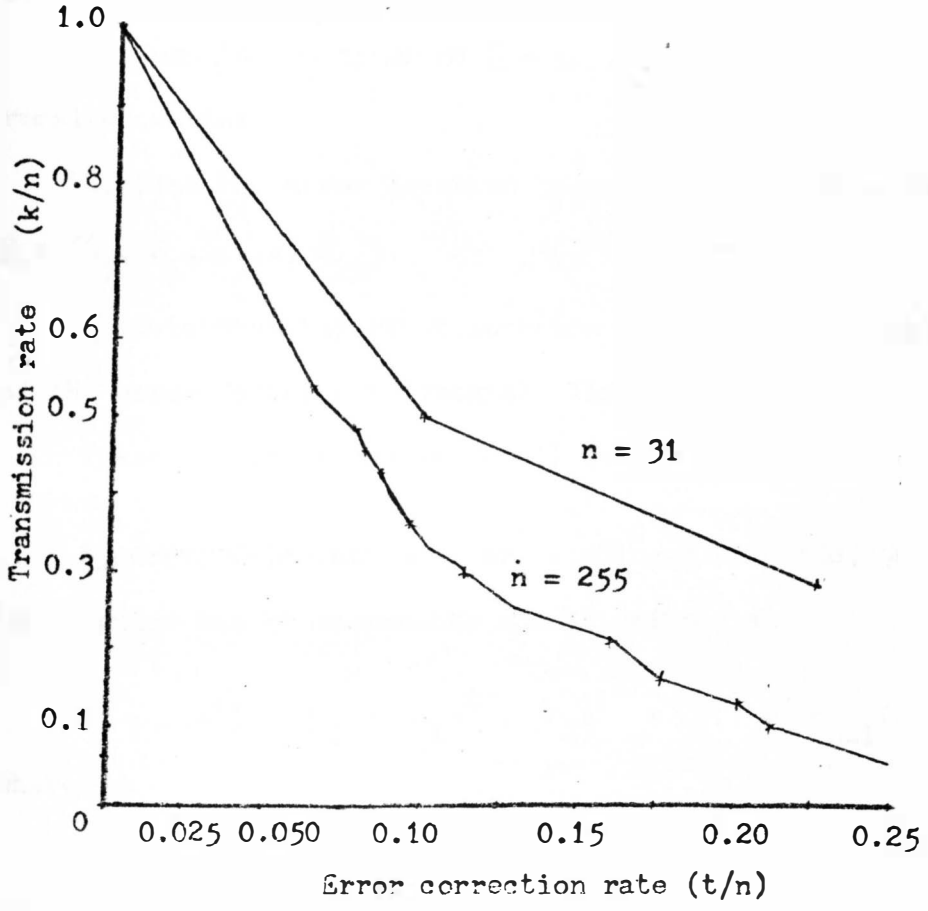


Figure 3.1.

Error-correcting capability of BCH codes.

The weight of this code vector is 7. Therefore, $d_{min} = 2t + 1 = 7$. To verify that the (15, 5) code described above is a cyclic code, we divide $(x^n + 1)$ by $g(x)$. If it is divisible, then this (15, 5) code is a BCH cyclic code.

3.3. Cyclic Decoding Procedure

The decoding procedure for a BCH binary cyclic code consists of three main steps:

1. Compute the syndrome $\underline{S} = (S_1, S_2, S_3 \ldots S_{2t})$ from the received vector.

2. Find the error location polynomial $\sigma(x)$ from the syndrome $\underline{S} = (S_1, S_2, \ldots, S_{2t})$.

3. Determine the error location numbers by finding the roots of the error location polynomial $\sigma(x)$.

These are described in detail in the following section.

Syndrome Calculation: For decoding a BCH code, the syndrome to be computed has 2t components and is defined as

$$S_i = r(a^i) = r_0 + r_1 a^i + r_2 a^{i^2} + \ldots + r_{n-1}(a^i)^{n-1}$$

where

$$i = 1, 2, \ldots 2t.$$

Now, if V(x) is the transmitted code vector and r(x) is the received code vector, then

$$e(x) = r(x) + v(x)$$

where e(x) is the error code vector. It follows then that

$$\underline{S}_i = \underline{V}(a^i) + \underline{e}(a^i)$$

But a^i (for i = 1, 2, ..., 2t) is the root of the code polynomial V(x). Therefore,

$$\underline{S}_i = \underline{e}(a^i)$$

If e(x) is assumed to be an error pattern with v errors,

$$e(x) = x^{j_i} + x^{j_2} + x^{j_v}$$

$$S_1 = a^{j_i} + a^{j_2} + ... + a^{j_v}$$

$$S_2 = a^{(j_1)^2} + a^{(j_2)^2} + ... + a^{(j_v)^2}$$

$$\vdots$$

$$S_{2t} = a^{(j_1)^{2t}} + a^{(j_2)^{2t}} + ... + a^{(j_v)^{2t}}$$

Once $(a^{j_1}, a^{j_2}, ..., a^{j_v})$ have been found, then the powers of $(j_1, j_2, ..., j_v)$ will give us the 2t syndrome components and the error locations in e(x).

The following procedure is an important and effective way of finding a^{j_1} from S_i. Let $B_l = a^{j_l}$ for $1 \leqslant l \leqslant v$ where B_l is the error location number. Then we have

$$S_1 = B_1 + B_2 + ... + B_v$$

$$S_2 = B_1^2 + B_2^2 + ... + B_v^2$$

$$\vdots$$

$$S_{2t} = (B_1)^{2t} + (B_2)^{2t} + (B_3)^{2t} + ... (B_v)^{2t}$$

These 2t components are symmetric functions[1] in $B_1, B_2, ..., B_v$ and are called power sum symmetric functions.

We define the error location numbers B_i such that

$$\sigma(x) = (1 + B_1x)\ (1 + B_2x)\ \ldots\ (1 + B_vx)$$

i.e. $\sigma(x) = \sigma_0 + \sigma_1 x + \sigma_2 x^2 + \ldots + \sigma_v x^v$

where

$$\sigma_0 = 1$$

$$\sigma_1 = B_1 + B_2 + \ldots + B_v$$

$$\sigma_2 = B_1B_2 + B_2B_3 + \ldots + B_{v-1}B_v$$

$$\vdots$$

$$\sigma_v = B_1B_2B_3 \ldots B_v$$

σ_i (i = 1,2, ..., v) are called elementary symmetric functions.[1] B_1^{-1}, B_2^{-1}, ... B_v^{-1} are roots of $\sigma(x)$, the error location polynomial and are inverses of the error location numbers.

As can be seen, the coefficients of $\sigma(x)$ are related to the syndrome components S_1, S_2, ..., S_{2t}, and it is therefore possible to find the error location polynomial $\sigma(x)$ from S_i. Once $\sigma(x)$ is found it is straightforward to find the error location numbers ($B_l = a^{t_l}$) and thus obtain the error pattern e(x).

3.4. Hardware Implementation

The method described earlier shows us that it is possible to compute $\sigma(x)$ from S_i, but does not indicate how to implement the computation. In a decoder design, this seems to be the most difficult step to implement.

Several papers have been presented to implement a circuit that would find $\sigma(x)$ from S_i. Among these, the following were most useful.

1. An alternative algorithm by Berlekamp.[2]
2. Peterson's[3] decoding procedure.

The first method has application for large t, i.e., ($t \geqslant 4$). These methods have been described at great length by both these men in their books.[2,3] However, when t is small ($t \leqslant 3$), it is easier to adopt a direct decoding procedure using simple combinational logic.[4]

A direct decoding procedure[4] with necessary theory is presented below. It might be pointed out, again, that the direct decoding method becomes very cumbersome for $t \geqslant 4$.

3.5. Direct Decoding Procedure[4]

As pointed out earlier, it is necessary to compute $\sigma(x)$ from S_i before the final error-correction step can be carried out. The direct method of decoding eliminates this step and as will be clear from the presentation below, it is much easier to implement for small t.

When applying the cyclic decoding procedure, it is only necessary to detect whether a 1 is a root of the error location polynomial $\sigma(x)$. In other words, suppose the decoder is to test the r_{n-1} digit. To do this, it tests whether a^{n-1} is an error location number, which is equivalent to testing whether a is a root of $\sigma(x)$. If a is a root, we have

$$\sigma_0 + \sigma_1 a + \sigma_2 a^2 + \ldots + \sigma_v a^v = 0$$

But $\sigma_0 = 1$.

Hence, $\sigma_1 a + \sigma_2 a^2 + \ldots + \sigma_v a^v = 1$.

i.e. $$\sum_{k=1}^{v} \sigma_k a^k = 1$$

To decode the r_{n-1} digit, the decoder therefore, forms $\sigma_1 a$, $\sigma_2 a^2$, $\ldots$ $\sigma_v a^v$. If the sum $\sigma_1 a + \sigma_2 a^2 + \ldots + \sigma_v a^v = 1$, then a^{n-1} is an error location number and r_{n-1} is an erroneous digit; otherwise r_{n-1} is a correct digit.

Now, the elementary symmetric functions σ_k are related to the power sum symmetric functions S_i by Newton's Identities:[10]

$$S_1 - \sigma_1 = 0$$

$$S_2 - S_1\sigma_1 + 2\sigma_2 = 0$$

$$S_3 - S_2\sigma_1 + S_1\sigma_2 - 3\sigma_3 = 0$$

$$S_4 - S_3\sigma_1 + S_2\sigma_2 - S_1\sigma_3 + 4\sigma_4 = 0$$

$$S_5 - S_4\sigma_1 + S_3\sigma_2 - S_2\sigma_3 + S_1\sigma_4 - 5\sigma_5 = 0$$

The first t odd power sum symmetric functions, S_i $(i = 1,\ldots,t)$, can be computed from the received word. The first t even ones can be found from the fact that in mod 2 algebra, $(a + b)^2 = a^2 + b^2$, and $S_{2i} = S_i^2$. Thus

$$S_1^2 = S_2$$

$$S_1^4 = S_4$$

$$S_3^2 = S_6, \text{ etc.}$$

In the above identities, σ_1, σ_2, ... σ_v are unknowns and $\sigma_k = 0$ for $k > t + 1$. This set of t linear equations can be solved and the unknown σ_k's can be written as functions of the known quantities, S_i's.

The above theory has been presented by Peterson and the theorem which implies this is as follows.[3]

Theorem:

The t x t matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ S_2 & S_1 & 1 & 0 & \cdots & 0 \\ S_4 & S_3 & S_2 & S_1 & \cdots & 0 \\ \vdots & & & & & \\ S_{2t-2} & S_{2t-3} & S_{2t-4} & S_{2t-5} & \cdots & S_{t-1} \end{bmatrix}$$

is non-singular if power-sum symmetric functions S_i are power sums of t or t - 1 distinct field elements, and is singular if the S_i are power sums of fewer than t - 1 distinct field elements.

In other words, $A \neq 0$ if S_i are power sums of t or t - 1 distinct roots, and $A = 0$ if S_i are power sums of t - 2 or fewer distinct roots. For proof, refer to Reference 6.

In matrix notation, this process can be carried out as follows.

$A\sigma = B$ (for binary case)

where

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ S_2 & S_1 & 1 & 0 & \cdots & 0 \\ S_4 & S_3 & S_2 & S_1 & \cdots & 0 \\ \vdots & & & & & \\ S_{2t-2} & S_{2t-3} & S_{2t-4} & S_{2t-5} & \cdots & S_{t-1} \end{bmatrix}$$

and

$$B = \begin{bmatrix} S_1 \\ S_3 \\ S_5 \\ \vdots \\ S_{2t-1} \end{bmatrix}$$

If the determinant $|A| \neq 0$, i.e., if the S_i are power sums of t, or t - 1 distinct roots, then

$$\sigma_k = \frac{1}{|A|} \sum_{i=1}^{t} S_{2i-1} A_{i,k} \quad k = 1, 2, \ldots, t$$

where $A_{i,k}$ (k = 1, 2, ..., t) are co-factors of the determinant $|A|$. Now, if 1 is a root of the polynomial $\sigma(x)$, then

$$\sum_{k=1}^{t} \sigma_k = 1.$$

It follows, therefore, that

$$\sum_{k=1}^{t} \frac{1}{|A|} \sum_{i=1}^{t} S_{2i-1} A_{i,k} = 1$$

Since A is independent of k,

$$\sum_{k=1}^{t} \sum_{i=1}^{t} S_{2i-1} A_{i,k} - |A| = 0.$$

Since we are working in a field of characteristic two, the equation above is equivalent to setting the determinant Δ = zero, where

$$\Delta = \begin{vmatrix} 1 & 1 & 1 & \cdots & 1 \\ S_1 & 1 & 0 & \cdots & 0 \\ S_3 & S_2 & 1 & \cdots & 0 \\ S_5 & S_4 & S_3 & \cdots & 0 \\ S_{2t} & S_{2t-2} & S_{2t-3} & \cdots & S_{t-1} \end{vmatrix}$$

i.e. $$= \begin{vmatrix} 1 & 1 \cdots 1 \\ B & A \end{vmatrix} = 0.$$

When $|A| = 0$, the last two equations in Newton's identities are deleted and we are left with (t - 2) equations and (t - 2) unknowns.

The above method is illustrated in the next chapter which deals with implementation of a (15, 7), 2-error correcting BCH cyclic code.

CHAPTER IV

THE CYCLIC CODE ENCODER

This chapter addresses the design of a cyclic code encoder in general, and a (15, 7) BCH cyclic code encoder in particular. An attempt has been made to simplify or eliminate cumbersome mathematical expressions and formulas and to present an easily understandable design procedure.

4.1. Generator Polynomial

Consider the Galois Field $GF(2^4)$, i.e., $m = 4$. We then have a code having block length n given by

$$n = 2^m - 1 = 2^4 - 1 = 15$$

corresponding to $m = 4$, we get the primitive polynomial $p(x)$ from the table of primitive polynomials* as $p(x) = 1 + x + x^4$.

Let a be a primitive element of $GF(2^4)$. The generator polynomial of a BCH cyclic code is given by

$$g(x) = \text{LCM}\ (M_1(x), M_3(x), \ldots., M_{2t-1}(x))$$

where $M_1(x), M_3(x), \ldots., M_{2t-1}(x)$ are the minimum polynomials of $a, a^3, \ldots., a^{2t-1}$ respectively. Suppose we want to design an encoder for a 2-error correcting BCH cyclic code. Then $t = 2$, and we have

$$g(x) = \text{LCM}\ (M_1(x), M_3(x))$$

* See Appendix A.

4.2. Minimum Polynomials*

To find the minimum polynomial, $M_1(x)$, consider a = B. Then we take powers of B^2, until a starts repeating:

$(B^2)^0 = a$

$(B^2)^1 = a^2$

$(B^2)^2 = a^4$

$(B^2)^3 = a^8$

$(B^2)^4 = a^{16} = a$ (repetition begins)

This means that $M_1(x)$ has a, a^2, a^4, and a^8 as roots; and therefore, by definition,

$$M_1(x) = (x + a)(x + a^2)(x + a^4)(x + a^8)$$
$$= x^4 + (a^8+a^4+a^2+a)x^3 + (a^{12}+a^{10}+a^9+a^6+a^5+a^3)x^2$$
$$+ (a^{14} + a^{13} + a^{11} + a^7)x + a^{15}$$

Substituting the values of a^i for i = 4, 5, 15, from Table A.2 we get

$$M_1(x) = x^4 + (a^2 + 1 + a + 1 + a^2 + a)x^3 + (a^3 + a^2 + a + 1$$
$$+ a^2 + a + 1 + a^3 + a^2 + a^3 + a + a^2 + a + a^3)x^2$$
$$+ (a^3 + 1 + a^3 + a^2 + 1 + a^3 + a^2 + a + a^3 + a + 1)x + 1$$

$$M_1(x) = x^4 + x + 1^{**}$$

In order to find $M_3(x)$, the minimum polynomial of a^3 we proceed in the same fashion, except that this time $B = a^3$

$(B^2)^0 = a^3$

* See Appendix A.

**Other methods of finding the minimum polynomial are given in References 2 and 3.

$(B^2)^1 = a^6$

$(B^2)^2 = a^{12}$

$(B^2)^3 = a^{14} = a^9$

$(B^2)^4 = a^{48} = a^3$ (repetition begins)

Therefore, $M_3(x)$ has a^3, a^6, a^9 and a^{12} as roots. Thus,

$$M_3(x) = (x + a^3)\ (x + a^6)\ (x + a^9)\ (x + a^{12}).$$

Simplifying and substituting the values of a, we get

$$M_3(x) = x^4 + x^3 + x^2 + x + 1.$$

Once we obtain $M_1(x)$, and $M_3(x)$, straightforward techniques allow us to find the generator polynomial, the number of information and parity check digits for the code under investigation, and the minimum distance of the code.

By definition,

$$g(x) = \text{LCM}\ (M_1(x),\ M_3(x))$$
$$= \text{LCM}\ ((x^4 + x + 1)\ (x^4 + x^3 + x^2 + x + 1))$$

Since both $M_1(x)$ and $M_3(x)$ are irreducible,

$$g(x) = (x^4 + x + 1)\ (x^4 + x^3 + x^2 + x + 1)$$

or $$g(x) = x^8 + x^7 + x^6 + x^4 + 1.$$

The degree of g(x), the generator polynomial will give us the number of parity check digits (n - k), i.e., (n - k) = 8. Therefore,

$$k = n - (n - k) = 15 - 8 = 7.$$

We have thus come to a point where we can conclude that the generator polynomial

$$g(x) = 1 + x^4 + x^6 + x^7 + x^8$$

generates a (15, 7) BCH cyclic code that can correct 2 (t = 2) or fewer random errors. The weight of the generator polynomial is defined as the number of binary 1's present in g(x). In this case the number of binary 1's = 5.

Hence, the weight of the generator polynomial = 5.

i.e., $d_{min} = 5$.

As a check we see that $d_{min} = 2t + 1 = 2.2 + 1 = 5$.

Therefore, this is a double-error correcting BCH cyclic code.

4.3. Encoder Design Consideration

While designing an encoder, a designer can either choose a k-stage encoder or an (n - k) stage encoder, depending on the parameters n and k of the code.

In general, if (n - k) > k, it is more economical to design a k-stage encoder, for it will need only a k-stage shift register. In our example, k = 7, n - k = 8, we therefore, would proceed to design a k-stage (k = 7) encoder.

4.3.1. K-Stage Encoder

A k-stage encoder for a cyclic code is simply a k-stage shift register with necessary feedback connections. This k-stage register is designed in such a way that any incoming polynomial is divided by the generator polynomial of the code and the remainder is stored in the storage devices, e.g. flip-flops. This remainder gives us the necessary parity check digits to form a codeword.

In order to design a k-stage encoder, we require the parity check polynomial h(x) given by

$$h(x) = h_0 + h_1x + h_2x^2 + ... + h_kx^k$$

If we know g(x), the generator polynomial, then the parity check polynomial, h(x), is given by

$$h(x) = \frac{x^n + 1}{g(x)}$$

In our example, $g(x) = 1 + x^4 + x^6 + x^7 + x^8$

Therefore,

$$h(x) = \frac{x^{15} + 1}{1 + x^4 + x^6 + x^7 + x^8} = 1 + x^4 + x^6 + x^7$$

This parity check equation or polynomial will tell us how to provide the feedback connections. We will use the following notations and symbols.

1. The symbol → FF → denotes a single binary shift register stage (a flip-flop), which is shifted by an external synchronous clock, so that its input at a particular time appears at its output one unit of time later.

2. The symbol → ⊕ → denotes an exclusive-OR gate.

3. The symbol h_i simply denotes a connection if $h_i = 1$; no connection if $h_i = 0$.

In our example k = 7. We would, therefore, need a 7-stage shift register. In general, for a k-stage encoder, we would need the following.

1. k - flip-flops.

2. At most (k - 1) exclusive-OR gates.

3. A counter and gates to control the feedback connections.

A general configuration for an encoder circuit is shown in Figure 4.1.

In our example:

$$h(x) = h_0 + h_4 x^4 + h_6 x^6 + h_7 x^7$$

We will have feedback connections at h_4 and h_6. This encoder circuit is shown in Figure 4.2.

As an example, suppose the message digits to be encoded are:

$$m = 1\ 0\ 1\ 0\ 0\ 1\ 0$$

The message polynomial is given by

$$m(x) = m_0 + m_1 x + m_2 x^2 + \ldots + m_{k-1} x^{k-1}$$

i.e. $$m(x) = m_0 + m_2 x^2 + m_5 x^5$$

The message block of k digits will be encoded into a codeword of n digits (n > k). Since n = 15, we require n - k = 8 parity check digits.

4.3.2. Encoding Procedure

The encoding procedure is as follows. Gate 1 is on, Gate 2 is off. The 7 information bits are shifted into the flip-flops serially, last digit first; simultaneously, they are shifted into the communication channel.

The register's contents at this stage are

1 0 1 0 0 1 0

Gate 1 is turned off now and Gate 2 turned on; the first parity check appears instantly at P, as in Figure 4.2. When an external clock

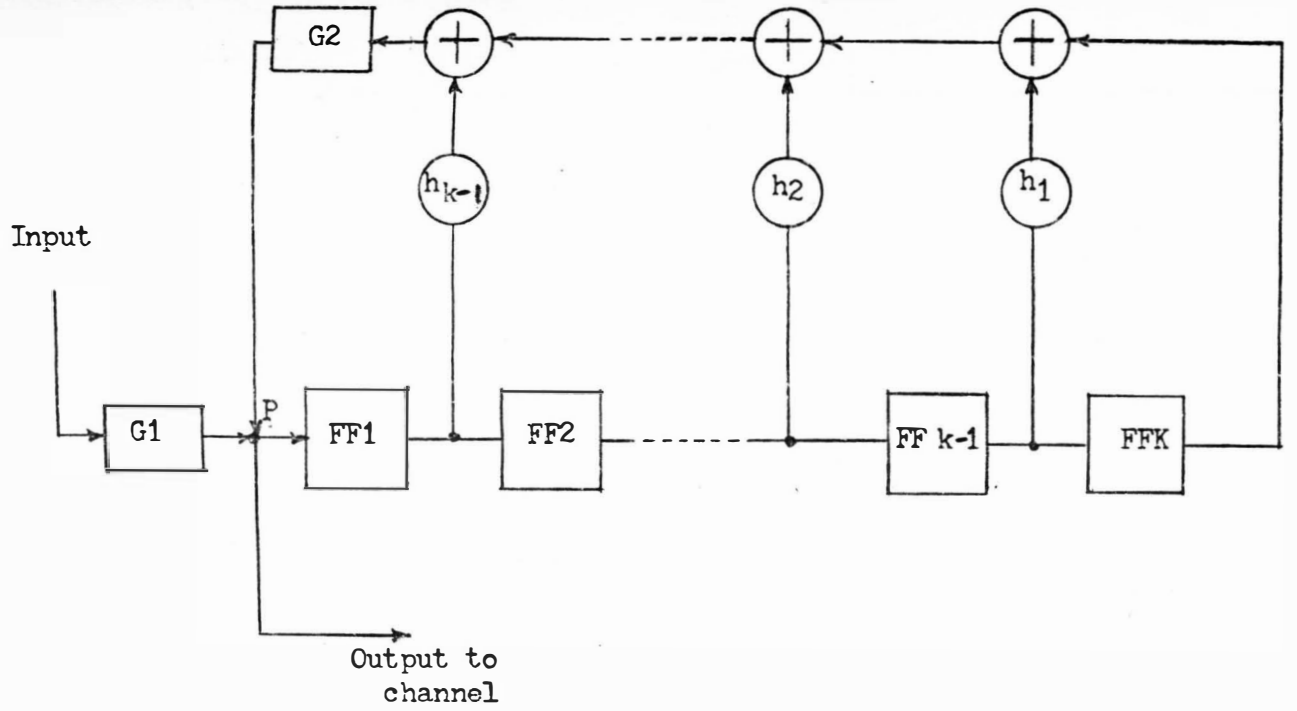


Figure 4.1. Cyclic code encoder.

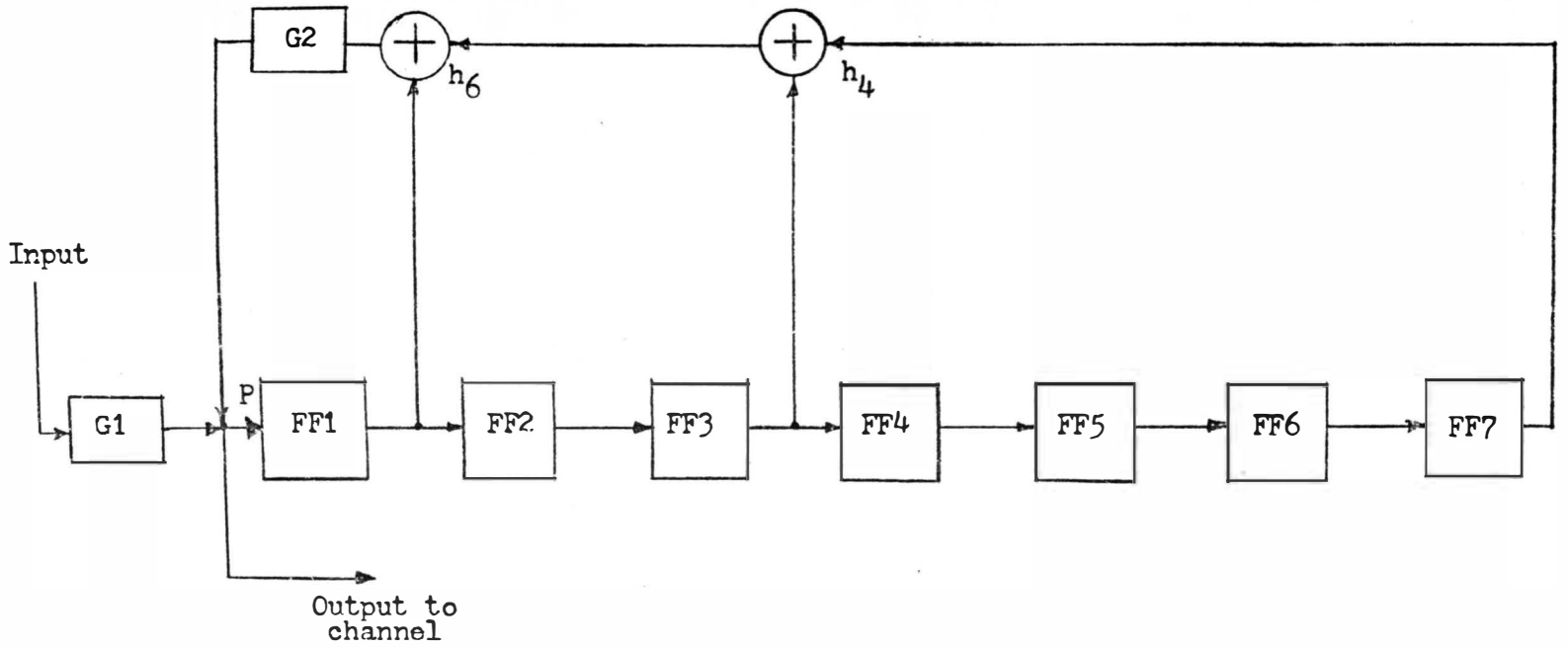


Figure 4.2. (15, 7) Cyclic code encoder.

pulse is applied, this parity check digit is shifted into the first flip-flop, ff1, and is also sent out to the channel.

Instantly at junction P, the second parity check digit appears. With the next clock pulse, this second parity check is shifted into the channel and also into the shift register; again, now the third parity check digit appears at P. This is continued until all the parity check digits are sent to the communication channel. The shift register is now ready for the next message block. The contents of the register at this stage give the remainder, resulting from dividing $x^{n-k}n(x)$ by $g(x)$, which are also the parity check digits. It is to be noted that for a k-stage encoder, there will be (n-k-k) bits missing, since there are only k stages of the shift register.

4.4. Encoder Analysis

It has been pointed out earlier that if the encoder is properly designed, it will divide the incoming polynomial (in systematic form) by the generator polynomial and the remainder will be the set of parity check digits.

In the following, we actually divide the incoming polynomial by the generator polynomial and note the resulting remainder. We also analytically determine the shift register contents after each pulse and see if the two results agree; this checks the design of the encoder.

The message polynomial is of the form $m(x) = m_0 + m_1x + m_2x^2 + \ldots + m_{k-1}x^{k-1}$.

In our example, $m(x) = 1 + x^2 + x^5$ or $m = 1\ 0\ 1\ 0\ 0\ 1\ 0$.

In order to make a systematic codeword, we multiply $m(x)$ by $x^{n-k} = x^8$ to obtain

$$m(x)\, x^{n-k} = x^{13} + x^{10} + x^8$$

In systematic form, the message polynomial is given by

$$m_0\, x^{n-k} + m_1\, x^{n-k+1} + \ldots + m_{k-1} x^{n-1}$$

The incoming message block in systematic form is, therefore, 1 0 1 0 0 1 0. Dividing the systematic codeword, $m(x)\, x^{n-k}$, by the generator polynomial, $g(x)$, we obtain the quotient polynomial $x^5 + x^4 + x + 1$, and the remainder is $r(x) = x^6 + x + 1$.

(Note $r(x) = r_0 + r_1 x + \ldots + r_{n-k-1} x^{n-k-1}$

Therefore, $= 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0$

This is the set of parity check digits. The entire codeword is given by

$$v(x) = r_0 + r_1 x + \ldots + r_{n-k-1} x^{n-k-1} + m_0 x^{n-k} + m_1 x^{n-k+1} + \ldots + m_{k-1} x^{n-1}$$

i.e., $v = 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0$

Parity bits | Info. bits

We now analyze the dividing circuit step by step and see if we get the same set of parity check digits.

We see from Table 4.1, that we get the same set of parity check digits, when we analyze the circuit step by step. Thus, the design is right and the encoder will encode all incoming messages into codewords to be transmitted over the communication channel.

Table 4.1. Encoder Register Contents

Shift no. t	Gate status G1	Gate status G2	Register content FF1	FF2	FF3	FF4	FF5	FF6	FF7	Symbol at P after t shift
Initial	on	off	0	0	0	0	0	0	0	0
1	on	off	0	0	0	0	0	0	0	1
2	on	off	1	0	0	0	0	0	0	0
3	on	off	0	1	0	0	0	0	0	0
4	on	off	0	0	1	0	0	0	0	1
5	on	off	1	0	0	1	0	0	0	0
6	on	off	0	1	0	0	1	0	0	1
7	off	on	1	0	1	0	0	1	0	0
8	off	on	0	1	0	1	0	0	1	1
9	off	on	1	0	1	0	1	0	0	0
10	off	on	0	1	0	1	0	1	0	0
11	off	on	0	0	1	0	1	0	1	0
12	off	on	0	0	0	1	0	1	0	0
13	off	on	0	0	0	0	1	0	1	1
14	off	on	1	0	0	0	0	1	0	1

CHAPTER V

THE CYCLIC CODE DECODER

The function of a decoder for a BCH cyclic code is to receive the transmitted message, decode it to decide if there is an error, and if an error is present, try to correct it. This chapter describes these functions of a decoder in detail and shows ways to implement the design.

5.1. Derivation of the Logical Equations

It has been shown in Chapter III that

$$A = \sigma B$$

where

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ S_2 & S_1 & 1 & \cdots & 0 \\ S_4 & S_3 & S_2 & \cdots & 0 \\ S_{2t-2} & S_{2t-3} & S_{2t-4} & \cdots & S_{t-1} \end{bmatrix} \qquad (5\text{-}1)$$

$$B = \begin{bmatrix} S_1 \\ S_3 \\ \cdot \\ \cdot \\ \cdot \\ S_{2t-1} \end{bmatrix}$$

t = number of errors to be corrected

σ = error location polynomial

It has been pointed out earlier that $A \neq 0$ whenever the power sums $S_1, S_2, \ldots, S_{t-1}$ are not identically zero. Consider the example of (15, 7) BCH cyclic code. To correct 2 or fewer errors, we have

$t = 2$

$m = 4$

$p(x)$ = primitive polynomial = $x^4 + x + 1$

Then, from Equation (5-1)

$$A = \begin{vmatrix} 1 & 0 \\ S_2 & S_1 \end{vmatrix} = S_1 \neq 0$$

Since $A \neq 0$,

$$= \begin{vmatrix} 1 & 1 & 1 \\ S_1 & 1 & 0 \\ S_3 & S_2 & S_1 \end{vmatrix} = S_1(1 + S_1 + S_1^2) + S_3 = 0 \qquad (5\text{-}2)$$

It is to be noted that S_2 has been replaced by S_1^2, since $S_{2i} = S_i^2$.

Now, S_1 and S_3 will be of the form

$$S_1 = a_0 + a_1 a + a_2 a^2 + a_3 a^3 \qquad (5\text{-}3)$$

$$S_3 = b_0 + b_1 a + b_2 a^2 + b_3 a^3$$

where a_i and b_i are binary coefficients, and

$i = 0, 1, 2, 3.$

Substituting these equations for S_1 and S_3 in (5-2) and making use of the relation $\bar{a} = 1 + a$, we obtain the following.

$$
\begin{aligned}
A &= a_0\bar{a}_2 + a_2\bar{a}_1 + a_1a_3 + b_0 = 0 \\
B &= (a_1 + a_2)\,\bar{a}_0 + a_3\bar{a}_2 + b_1 = 0 \\
C &= (a_0 + a_1)\,(a_1 + a_2 + a_3) + a_3\bar{a}_2 + b_2 = 0 \\
D &= (a_1 + a_2)\,\bar{a}_3 + a_3 + b_3 = 0
\end{aligned}
\qquad (5\text{-}4)
$$

The + sign denotes Ring-sum in Equations (5-3) and (5-4). From these relations we design the correcting unit. They can easily be implemented using Nand, not and Exclusive-OR gates.

5.2. The Decoding Procedure

The decoding procedure, now simplifies to the following steps.

1. Compute the power sums S_1, S_3 from the received vector.

2. Decode the received vector on a bit-by-bit basis, decoding higher-order bits first. Thus, to decode the r_{n-1} bit, the decoder tests whether r^{n-1} is an error location number, i.e., whether a is a root of $\sigma(x)$. This is done by taking the mod 2 sum

$$S_1a + S_2a^2 + \ldots + S_{2t}a^{2t}.$$

If the sum = 1, then a^{n-1} is an error location number and r_{n-1} is an erroneous digit.

3. If the sum = 1, this is added to the outgoing bit from the buffer; the bit is corrected and the message interpreted. This is continued until all the bits are checked and corrected of error, if any. The general configuration of the decoding circuit is shown in Figure 5.1. In the figure, the S_1 and S_3 blocks represent circuits to compute the power sums S_1 and S_3. The blocks, a and a^3, represent circuits for multiplying S_1 and S_3 by a and a^3, respectively. These

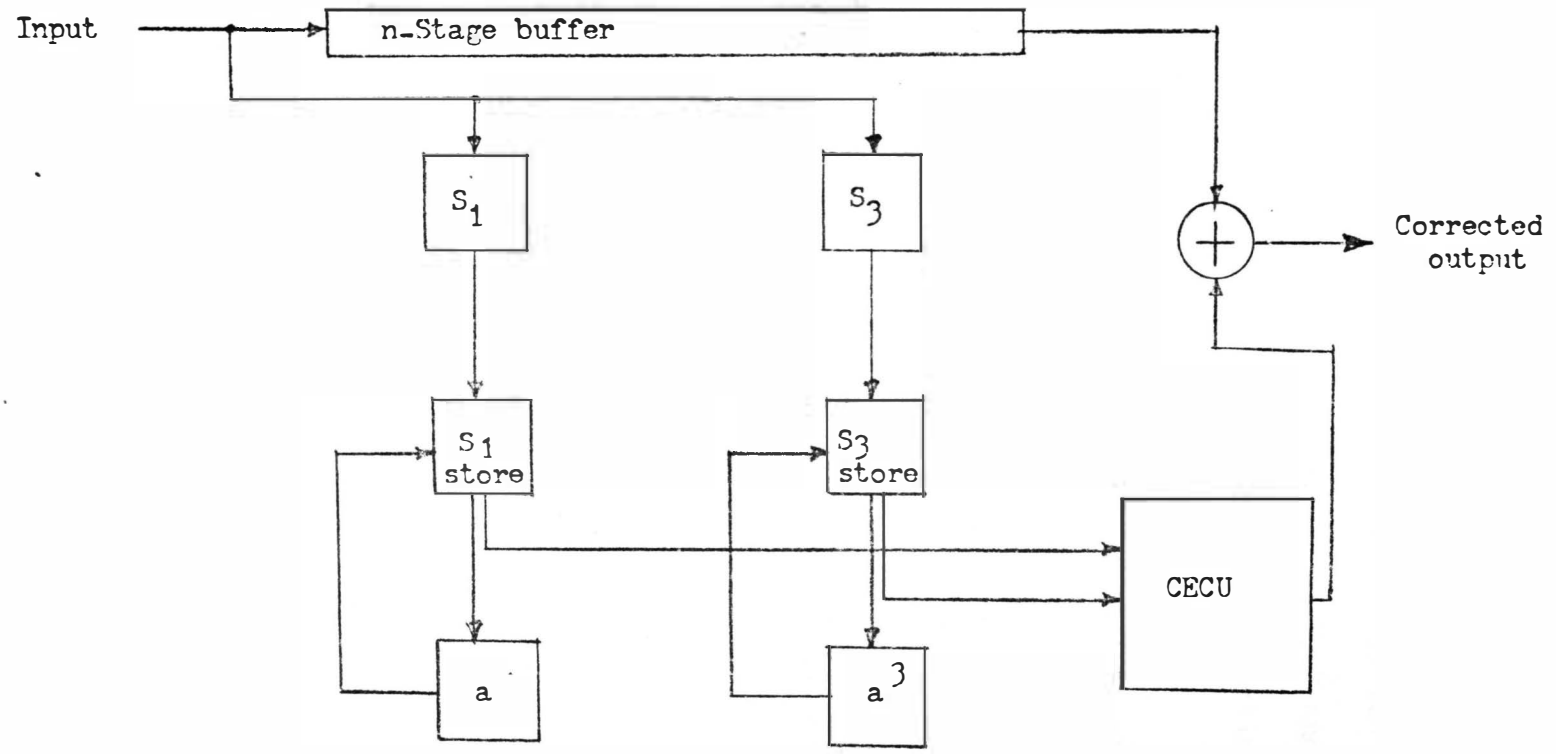


Figure 5.1

Cyclic code decoder

products, $S_1 a$ and $S_3 a$, are fed back and stored in the S_1 and S_3 registers; also they are sent to the cyclic error correcting unit to correct an error in the bit under investigation, if necessary.

During the next pulse, $S_1 a$ and $S_3 a^3$ stored in the top registers are multiplied by a and a^3 again and stored as $S_1 a^2$ and $S_3 a^6$. These are used to check the next bit. This process is continued until all the 15 bits have been checked.

In the following, each circuit of Figure 5.1 is explained in detail.

5.3. S_1 Register Design

The S_1 register or the shift register circuit which computes the power sum symmetric function S_1, is essentially a dividing circuit designed according to the minimum polynomial of a. To design the circuit we proceed as follows.

The parity check matrix for a 2-error correcting code is given by

$$\underline{H} = \begin{bmatrix} 1 & a & a^2 & 3 & 4 & \cdots & a^{n-1} \\ 1 & a^3 & a^6 & a^9 & a^{12} & \cdots & a^{3(n-1)} \end{bmatrix}$$

In our example, m = 4, t = 2, n = 15. Therefore, we have,

$$\underline{H} = \begin{bmatrix} 1\,0\,0\,0\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1 \\ 0\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1\,1\,0\,0 \\ 0\,0\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1\,1\,0 \\ 0\,0\,0\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1\,1 \\ 1\,0\,0\,0\,1\,1\,0\,0\,0\,1\,1\,0\,0\,0\,1 \\ 0\,0\,0\,1\,1\,0\,0\,0\,1\,1\,0\,0\,0\,1\,1 \\ 0\,0\,1\,0\,1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,1 \\ 0\,1\,1\,1\,1\,0\,1\,1\,1\,1\,0\,1\,1\,1\,1 \end{bmatrix} \qquad (5\text{-}5)$$

Now, to compute S_1, the dividing circuit will first divide the incoming n-digit word by the minimum polynomial of a, $M_1(x)$ and then the remainder is fed to the exclusive-or gates to get S_1. In the example the minimum polynomial, $M_1(x)$, is

$$M_1(x) = 1 + x + x^4.$$

To find the wired connection for the exclusive-OR gates required to compute S_1, we consider the first m (m = 4) columns of the H-matrix in (5-5)

$$1 \quad a \quad a^2 \quad a^3 \iff \begin{matrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{matrix}$$

It is apparent that each row has only one 1. Consequently, each flip-flop of the S_1-storage register has only one input and hence we do not need any exclusive-OR gates. The circuit diagram is shown in Figure 5-2.

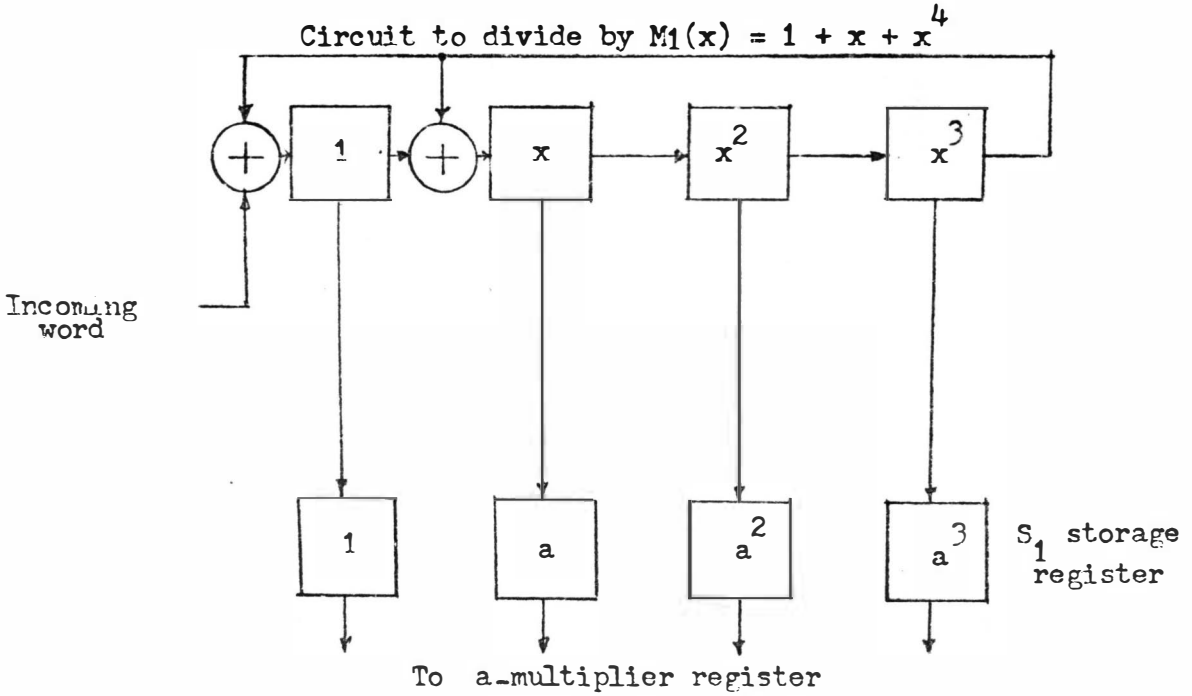


Figure 5.2

Circuit to compute S_1

To design the S_3 register, we consider the second m columns of the H-matrix,

$$1\ a^3\ a^6\ a^9 \qquad \begin{matrix} 1\ 0\ 0\ 0 \\ 0\ 0\ 0\ 1 \\ 0\ 0\ 1\ 0 \\ 0\ 1\ 1\ 1 \end{matrix}$$

The dividing circuit will divide the incoming polynomial by $M_3(x)$, the minimum polynomial of a^3. The last row of the above matrix has three 1's and hence requires 3 inputs to the S_3 storage register. A block diagram of the circuit is shown in Figure 5.3.

After all 15 digits of the first message block (incoming word) have been shifted into the $M_1(x)$ and $M_3(x)$ registers, the $M_1(x)$ and $M_3(x)$ registers are left with the remainders $r_1(x)$ and $r_3(x)$ respectively, where

$$r_1(x) = r(x)/M_1(x)$$

$$r_3(x) = r(x)/M_3(x)$$

These remainders when made to go through the Exclusive-OR gates, give S_1 and S_3.

5.4. Circuit Analysis

In the following, we will analyse the circuit designed earlier and check for its connections, by illustrating with the help of an example.

Referring back to Chapter IV, Section 4.4, the encoded word we consider is

$$v = 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0$$

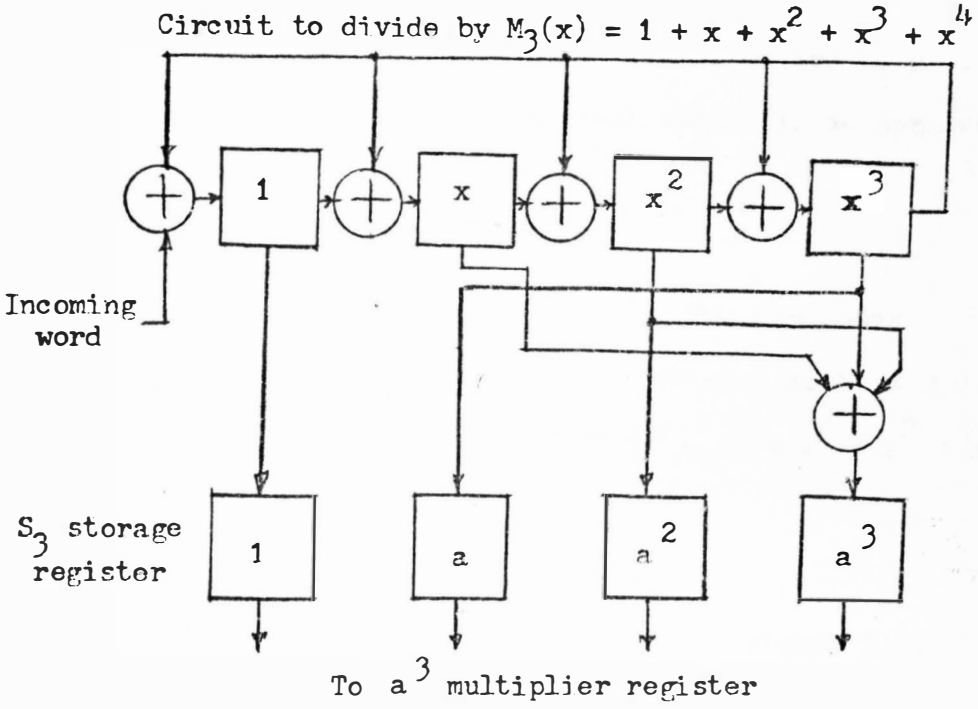


Figure 5.3

Circuit to compute S_3

Suppose, during the transmission of this message, errors are introduced and the word is changed to

$$r = 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1$$

As explained earlier if there is an error, or errors, S_1 and $S_3 \neq 0$, i.e., the remainders in the $M_1(x)$ and $M_3(x)$ registers will be non-zero.

The received polynomial is now

$$r(x) = 1 + x + x^6 + x^8 + x^{10} + x^{14}$$

To find S_1, we divide $r(x)$ by $M_1(x) = 1 + x + x^4$. The remainder is $r_1(x) = x^2$ which is stored in the S_1 storage register as $S_1 = 0\ 0\ 1\ 0$. Similarly, to get S_3 we first divide $r(x)$ by $M_3(x) = x^4 + x^3 + x^2 + x+1$. The remainder is $r_3(x) = 1 + x + x^2$

or $r_3 = 1\ 1\ 1\ 0$

After going through the Exclusive-OR gates as shown in Figure 5.3,

$$S_3 = 1\ 0\ 1\ 0$$

We now analyse the circuit contents after each shift and see if the reminders r_1 and r_3 agree with the contents after 15 shifts. The Table 5.1 shows the contents of the $M_1(x)$ and $M_3(x)$ registers after each shift. After the last digit is shifted into the registers $M_1(x)$ and $M_3(x)$, the contents are 0 0 1 0 and 1 1 1 0, respectively. These contents agree with the remainders obtained by actual divisions earlier. Now we check the Exclusive-OR gate connections. The received word is

$$r(x) = 1 + x + x^6 + x^8 + x^{10} + x^{14}$$

Table 5.1. $M_1(x)$ and $M_3(x)$ Register Contents

Incoming word	Register contents $M_1(x)$				Register content $M_3(x)$			
	1	a	a^2	a^3	1	a	a^2	a^3
Initial	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	1
1	0	1	0	0	0	1	1	1
0	0	0	1	0	1	1	0	0
1	1	0	0	1	1	1	1	0
0	1	0	0	0	0	1	1	1
1	1	1	0	0	0	1	0	0
0	0	1	1	0	0	0	1	0
0	0	0	1	1	0	0	0	1
0	1	1	0	1	1	1	1	1
0	1	0	1	0	1	0	0	0
1	1	1	0	1	1	1	0	0
1	0	0	1	0	1	1	1	0

Then, the power sum symmetric function S_1 is

$$S_1 = r(a) = 1 + a + a^6 + a^8 + a^{10} + a^{14}$$

Substituting the value of a^i for i = 4, 5, ... 15, from Table A.2, we have

$$S_1 = 1 + a + a^3 + a^2 + a^2 + 1 + a^2 + a + 1 + a^3 + 1$$

i.e.

$$S_1 = a^2 = 0\ 0\ 1\ 0$$

Similarly,

$$S_3 = r(a^3)$$
$$= 1 + (a)^3 + (a^6)^3 + (a^8)^3 + (a^{10})^3 + (a^{14})^3$$
$$= 1 + a^3 + a^3 + a^3 + a + 1 + a^3 + a^2 + a + 1$$

i.e.

$$S_3 = 1 + a^2 = 1\ 0\ 1\ 0$$

The values of S_1 and S_3 obtained here agree with the values obtained earlier.

5.5 a-Multiplier Circuit

After S_1 and S_3 have been computed, the next step is to multiply them by a and a^3. After S_1 is multiplied by a, the product is fed back to the S_1 storage register. Then, S_1a will be sent to the combinational logic to correct errors, if any. Similarly, S_3a^3 will be sent to the combinational circuit for the same purpose. Next, S_1a and S_3a^3 are multiplied by a and a^3, respectively. This process is continued until we obtain $S_1\ a^{2m-2}$ and $S_3\ a^{3(2^m-2)}$ in the storage

registers, which indicates that all the bits have been sent to the correction unit and checked for errors.

The design of the a-multiplier circuit is straightforward and easy to implement. From Table A.3, Appendix A, we obtain the primitive polynomial,

$$p(x) = 1 + x + x^4$$

Since we want to multiply by a, let

$$A = a$$

We find the product Aa^i for i = 0, 1, 2, ..., (m - 1). When m = 4,

$$Aa^0 = a = 0\ 1\ 0\ 0$$

$$Aa^1 = a^2 = 0\ 0\ 0\ 1$$

$$Aa^2 = a^3 = 0\ 0\ 0\ 1$$

$$Aa^3 = a^4 = a + 1 = 1\ 1\ 0\ 0$$

The circuit design is dependent on the following matrix.

$$Z_4, Z_3, Z_2, Z_1 = S_1 \begin{bmatrix} Aa^3 \\ Aa^2 \\ Aa^1 \\ Aa^0 \end{bmatrix}$$

$$= S_1 \begin{bmatrix} 1\ 1\ 0\ 0 \\ 0\ 0\ 0\ 1 \\ 0\ 0\ 1\ 0 \\ 0\ 1\ 0\ 0 \end{bmatrix}$$

We see that the first, third and fourth columns have only one 1 each and hence no need for Exclusive-OR gates. The second column has two

inputs and hence we need one Exclusive-OR gate. The circuit is shown in Figure 5.4.

5.6. a^3-Multiplier Circuit

The a^3-multiplier circuit is obtained using the same design rule as the a-multiplier circuit. Here

$$A = a^3$$

$$Aa^0 = a^3 = 0\ 0\ 0\ 1$$

$$Aa^1 = a^4 = a + 1 = 1\ 1\ 0\ 0$$

$$Aa^2 = a^5 = a^2 + a = 0\ 1\ 1\ 0$$

$$Aa^3 = a^6 = a^3 + a^2 = 0\ 0\ 1\ 1$$

$$Z_4, Z_3, Z_2, Z_1 = S_3 \begin{bmatrix} Aa^3 \\ Aa^2 \\ Aa^1 \\ Aa^0 \end{bmatrix} = S_3 \begin{bmatrix} 0\ 0\ 1\ 1 \\ 0\ 1\ 1\ 0 \\ 1\ 1\ 0\ 0 \\ 0\ 0\ 0\ 1 \end{bmatrix}$$

The circuit is shown in Figure 5.5.

Example:

Let $S_1 = 0\ 0\ 1\ 0 = a^2$

$S_1 a = aa^2 = a^3 = 0\ 0\ 0\ 1$

$S_3 = 1 + a^2 = 1\ 0\ 1\ 0$

$S_3 a^3 = a^3(1 + a^2) = a^3 + a^2 + a = 0\ 1\ 1\ 1$

Note that these values of $S_3 a^3$ and $S_1 a$ are obtained in the designed circuit. Therefore, the design is correct.

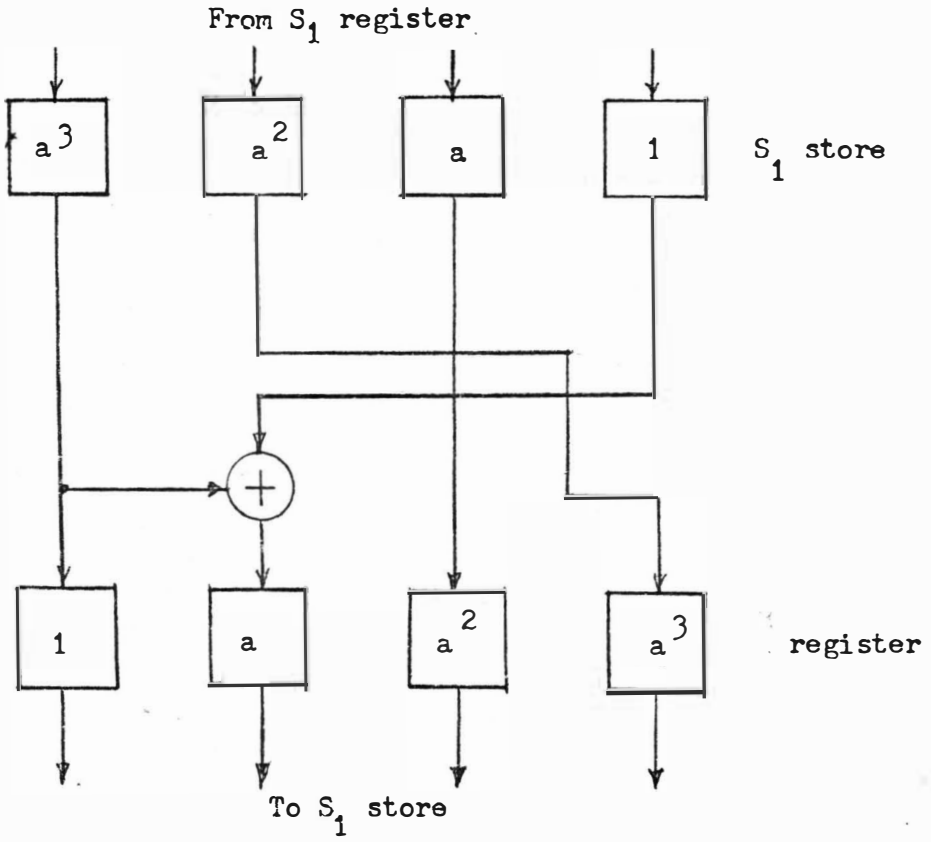


Figure 5.4

a -Multiplier circuit

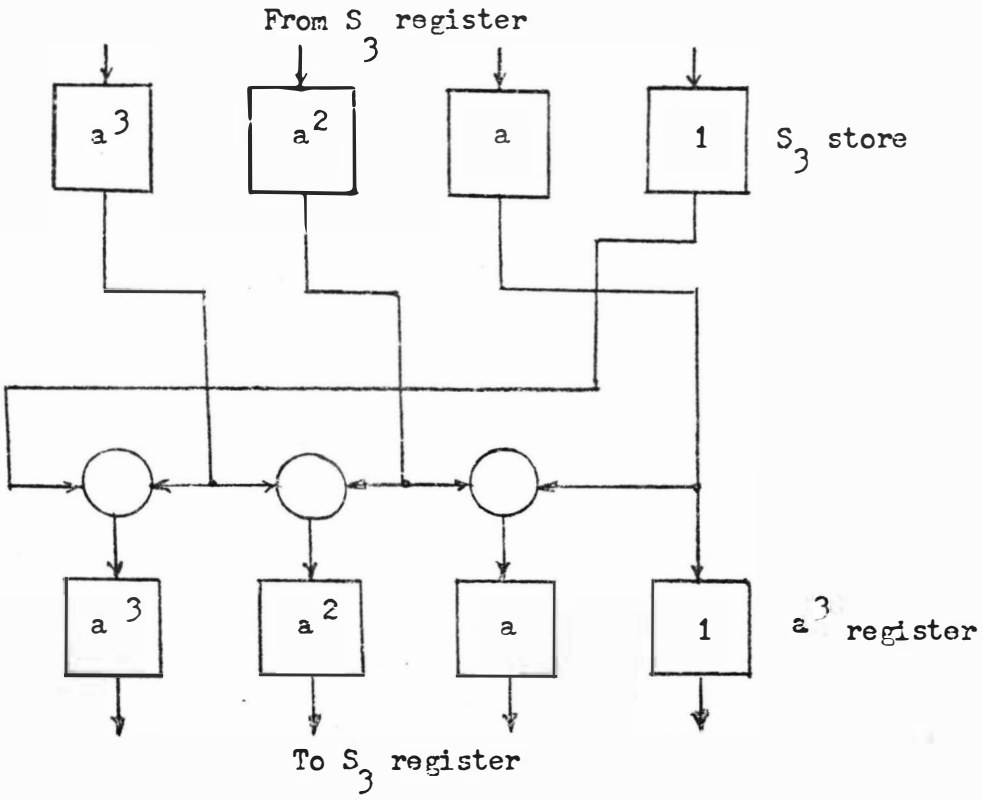


Figure 5.5

a^3 Multiplier circuit

5.7. The Cyclic Error Correction Unit

The error-correction unit is designed according to the set of four Boolean expressions, Equation 5-4, arrived at earlier in this chapter. As shown earlier, this unit has as its input the coefficients of $S_i a^i$ and $S_{3i} a^{3i}$ which are binary digits. The following example will help understand its working.

Suppose the r_{n-1} digit is to be checked for error. For this we need the coefficients of $S_1 a$ and $S_3 a^3$.

$S_1 a = 0\ 0\ 0\ 1$; $S_3 a^3 = 0\ 1\ 1\ 1$

since $S_1 = a_0 + a_1 a + a_2 a^2 + a_3 a^3$

and $S_3 = b_0 + b_1 a + b_2 a^2 + b_3 a^3$

$a_0 = 0,\ a_1 = 0,\ a_2 = 0,\ a_3 = 1$

$b_0 = 0,\ b_1 = 1,\ b_1 = 1,\ b_3 = 1$

Then, from Equation 5-4,

$$A = a_0 a_2 + a_2 a_1 + a_1 a_3 + b_0 = 0 + 0 + 0 + 0 = 0$$

$$B = (a_1 + a_2)\,\bar{a}_0 + a_3 \bar{a}_2 + b_1 = 0 + 1 + 1 = 0$$

$$C = (a_0 + a_1)\,(a_1 + a_2 + a_3) + (a_3 \bar{a}_2) + b_2 = 0 + 1 + 1 = 0$$

$$D = (a_1 + a_2)\,\bar{a}_3 + a_3 + b_3 = 0 + 1 + 1 = 0$$

$$\overline{A+B+C+D} = \overline{0 + 0 + 0 + 0} = 1$$

This means that the r_{n-1} digit is in error and is to be corrected. The correction is achieved by adding $\overline{A + B + C + D}$ to the r_{n-1} digit. Similarly, for correcting the r_{n-2} digit, we will need the coefficients

of $S_1 a^2$ and $S_3 a^6$. This means we check each digit on a bit-by-bit basis and correct it as soon as an error is found before the next digit is checked.

The following table gives the contents of the shift register during the correction procedure. As we can see the sequence of register contents starts repeating after $S_1 a^{2m-2}$, i.e., after all the 15 digits have been checked. The S_1 and S_3 registers are now ready to process the next message block. The circuit for this cyclic error correction unit is given in Appendix C. The actual operation of the encoder and the decoder, together with the functioning of the correcting unit, are discussed in detail in Appendix C.

Table 5.2. The S_1 and S_3 register contents.

	S_1 register					S_3 register			
S_1	a	0	0	0	1	S_3 a^3	0	1	1	1
S_1	a^2	1	1	0	0	S_3 a^6	1	0	0	1
S_1	a^3	0	1	1	0	S_3 a^9	0	0	1	0
S_1	a^4	0	0	1	1	S_3 a^{12}	0	1	1	0
S_1	a^5	1	1	0	1	S_3 a^{15}	1	0	1	0
S_1	a^6	1	0	1	0	S_3 a^3	0	1	1	1
S_1	a^7	0	1	0	1	S_3 a^6	1	0	0	1
S_1	a^8	1	1	1	0	S_3 a^9	0	0	1	0
S_1	a^9	0	1	1	1	S_3 a^{12}	0	1	1	0
S_1	a^{10}	1	1	1	1	S_3 a^{15}	1	0	1	0
S_1	a^{11}	1	0	1	1	S_3 a^3	0	1	1	1
S_1	a^{12}	1	0	0	1	S_3 a^6	1	0	0	1
S_1	a^{13}	1	0	0	0	S_3 a^9	0	0	1	0
S_1	a^{14}	0	1	0	0	S_3 a^{12}	0	1	1	0
S_1	a^{15}	0	0	1	0	S_3 a^{15}	1	0	1	0

CHAPTER VI

CONCLUSIONS

A (15, 7) BCH cyclic code encoder and decoder were designed and built using medium scale integrated circuit modules. The following IC's were required for building the device.

Flip-flops: SN 7476, JK Master-Slave type, dual

Nand gates: SN 7400, too-input, quadruple

Inverters: SN 7404, hex.

Exclusive-OR gates: SN 74L86, two-input, quadruple

The advantages of using gates and flip-flops in integrated circuit form are indeed manifold. For example, the SN 7476 flip-flop or shift register has 21 transistors, 3 diodes and 20 resistors. All these components occupy less than 1/2 $in.^2$ of surface area. Further, all we are required to do is to connect 8 pins. The increase in reliability and decrease in wiring and assembly cost of the system are apparent. In fact, in most cases it would have been impractical to build the unit with discrete components.

The decoder has a random-error-correcting capability of two or fewer errors. The efficiency of data transmission in this case is 7/15 or 46.66 percent. For most practical purposes, BCH cyclic code decoders are designed only to detect errors. In fact, that is one of the reasons for the lower data transmission efficiency in our decoder. The decoder becomes very versatile if it is designed only to detect

errors. A (15, 7) BCH cyclic code decoder designed only to detect errors will:

1. Detect all single, double, triple and quadruple errors in a message block.

2. Detect all burst errors consisting of a burst of length (n - k), i.e. 8 or less.

3. In addition, it will detect burst errors of length 9 or 10 in over 90 percent of the cases.

Further, since no error correction is desired, the need for the a-multiplier, the a^3-multiplier and the cyclic error correction unit does not exist, resulting in great savings in hardware and cost. Also, the time to process the message is considerably reduced; although this is partially offset by the time required to retransmit the erroneous data.

It might be of interest to those desiring to do research in this field, that as of today, convolutional codes have been established to be more efficient and easier to implement for error correction than BCH cyclic codes. There are several BCH cyclic codes more efficient than the (15, 7) code used for this research, especially the (31, 26) BCH cyclic code. This code has an efficiency of 83.9 percent and can be expected to be better than 99.9 percent effective in detecting errors.[5]

An advanced type of BCH cyclic code decoder has recently been built at the Lincoln Laboratory, M.I.T., by Baxter and Schneider.[13] It is a (127, 92) decoder which can correct 5 or fewer random errors

and detect all occurrences of 10 or fewer errors. The decoder is about the size of a desk-size computer and a 127-stage shift register. An experimental error correcting decoder has been built at the Bell Telephone Laboratory that can correct bursts as long as 1000 bits. The decoder utilizes generalized burst-trapping error control technique and its performance is being evaluated. For further details, Reference 14 is suggested.

REFERENCES

1. Lin Shu, An Introduction to Error-Correcting Codes, Prentice-Hall, Inc., New Jersey, 1970.

2. Berlekamp, Elwyn R., Algebraic Coding Theory, McGraw-Hill Book Co., New York, 1968.

3. Peterson, W. W., Error-Correcting Codes, M.I.T. Press, Cambridge, Massachusetts, 1961.

4. Chien, R. T., "Cyclic Error-Correcting Codes for Bose-Chaudhury Hocquenghem Codes," IEEE Transactions on Information Theory, Vol. IT-10, October, 1964.

5. Wilson, R. A., "Data Format and Error Control Relative Effective Efficiency," 1970 Midwest Power System Conference and Power Symposium, 1970.

6. Peterson, W. W., "Encoding and Error-Correction Procedures for the BCH Codes," IRE Transaction on Information Theory, September, 1960.

7. Birkhoff, G., and MacLane, S., A Survey of Modern Algebra. The MacMillan Co., New York, 1967.

8. Bose, R. C. and Chaudhury, D. K., "On a Class of Error-Correcting Binary Codes," Information and Control, Vol. 3, No. 1, March, 1960.

9. Bose, R. C. and Chaudhury, D. K., "Further Results on Error-Correcting Binary Group Codes," Information and Control, Vol. 3, No. 3, September, 1960.

10. Forney, G. David, Concatenated Codes, Research Monograph #37, M.I.T. Press, Cambridge, Massachusetts, 1966.

11. Abramson, N., Information Theory and Coding, McGraw-Hill Book Co., New York, 1963.

12. Burton, H. D., "Inversionless Decoding of BCH Codes," IEEE Transactions on Information Theory, Vol. IT-17, July, 1971.

13. Bartee, Thomas C. and Schneider, David I., "An Electronic Decoder for BCH Error-Correcting Codes," Lincoln Laboratory, M.I.T., Lexington 73, Massachusetts.

14. Pehlert, W. K. Jr., "Design and Evaluation of a Generalized Burst-Trapping Error Control System," Bell Telephone Laboratory, Holmdel, New Jersey, (unpublished report).

15. Albert, A. A., Fundamental Concepts of Higher Algebra, University of Chicago Press, Chicago, Illinois, 1956.

APPENDIX A

GALOIS FIELD ARITHMETIC AND ALGEBRA

This section provides an elementary knowledge of the algebra that is required to understand the cyclic code theory. The treatment is basically descriptive and not mathematically rigorous. More advanced treatments are contained in the reference list at the end of this paper.

Galois Field Arithmetic

It is possible to define addition and multiplication for a finite number of symbols, if the number of symbols is a power to a prime number* such that most of the rules of ordinary arithmetic apply. It is, therefore, possible to utilize most of the techniques of algebra.

For digital computers and for digital data transmission, we generally use two symbols, 0 and 1, for which additions and multiplication can be defined as follows.

Table A.1. Addition and multiplication in GF(2).

Addition	Multiplication
0 + 0 = 0	0.0 = 0
0 + 1 = 1	0.1 = 0
1 + 0 = 1	1.0 = 0
1 + 1 = 0	1.1 = 1

* A prime number p is an integer > 1, such that it is divisible only by $\pm$ p or $\pm$ 1. E.g. 2, 3, 5, 7, 11, 13,.... are prime numbers.

The addition and multiplication defined above are called the "modulo-2 addition and multiplication" respectively. The alphabet of two symbols, 0 and 1, together with the modulo-2 addition and multiplication is called a FIELD of two elements or a BINARY FIELD, usually denoted by GF(2). Note that since $1 + 1 = 0$, $1 = -1$.

Galois Field Algebra

To illustrate how the ideas of algebra can be used with the above arithmetic, we consider the following set of equations,

$$x + y = 1$$

$$x + z = 0$$

$$x + y + z = 1$$

We can solve these three equations by ordinary algebraic method to get $x = 0$, $y = 1$, $z = 0$. Since we are able to solve these equations, they must be Linearly Independent, and the determinant of the coefficient on the left side must be non-zero. If the determinant is non-zero, it must be 1. We can verify this as follows.

$$\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix} = 1 \begin{vmatrix} 0 & 1 \\ 1 & 1 \end{vmatrix} - 1 \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix} + 0 \begin{vmatrix} 1 & 0 \\ 1 & 1 \end{vmatrix}$$

$$= 1.1 - 1.0 + 1.1 = 1$$

Now consider the polynomials whose coefficients are either 0 or 1. For real numbers, if λ is a root of the polynomial $f(x)$, $f(x)$ is divisible by $(x - \lambda)$. This holds true in the case of $f(x)$ with binary coefficients.

For example, let

$$f(x) = x^4 + x^3 + x^2 + 1$$

Then $f(1) = 1^4 + 1^3 + 1^2 + 1 = 0$. $(\lambda = 1)$

Thus, $f(x)$ should be divisible by $(x-1) = (x+1)$.

$$(x^4 + x^3 + x^2 + 1) \div (x + 1) = (x^3 + x + 1)$$

The only polynomials of degree 1 are x and x + 1. The polynomials of degree 2 are x^2, $x^2 + x$, $x^2 + 1$ and $x^2 + x + 1$. Of these four polynomials, all but $x^2 + x + 1$ has either 0 or 1 as a root and hence divisible by x or (x + 1). However, $x^2 + x + 1$ does not have 0 or 1 as roots and so is not divisible by any polynomial except 1 and itself. A polynomial p(x) of degree m is said to be irreducible over the binary field GF(2) if p(x) is not divisible by any polynomial of degree less than m and greater than zero. Thus $x^2 + x + 1$ is an irreducible polynomial; and so is $x^4 + x + 1$. This can be verified by attempting to divide $x^4 + x + 1$ by all polynomials of degree less than m (m = 4) and greater than 1.

Galois Field with (2^m) Symbols

Fields with 2^m symbols are called Galois Field, $GF(2^m)$ and are very useful in the study of cyclic codes. In the following, a method to derive an arithmetic with 2^m symbols is described:[1]

First, we start with an arithmetic with two symbols, and a polynomial p(x) of degree m. Next we introduce a new symbol a, and assume that p(a) = 0, just as we may assume that 2 = 0 in arithmetic with two symbols. Then a table of powers of a is developed. If p(x) is chosen properly, the powers of a up to 2^m-2 will all be

different. The set of 2^m field elements will be 0, 1, a, a^2, ... a^{2m-2}, a^{2m-1}; and $a^{2^m-1} = 1$. In addition it is now possible to express each element of the field as a sum of the elements 1, a, a^2,.... a^{m-1}.

Example: Let $m = 4$, $p(x) = x^4 + x + 1$. Then we have the following table.

Table A.2. The set of 2^4 field elements.

0

1

a

a^2

a^3

$a^4 = a + 1$

$a^5 = a(a + 1) = a^2 + a$

$a^6 = a(a^2 + a) = a^3 + a^2$

$a^7 = a(a^3 + a^2) = a^4 + a^3 = a^3 + a + 1$

$a^8 = a(a^3 + a + 1) = a^4 + a^2 + a = a^2 + a + a + 1 = a^2 + 1$

$a^9 = a(a^2 + 1) = a^3 + a$

$a^{10} = a(a^3 + a) = a^4 + a^2 = a^2 + a + 1$

$a^{11} = a(a^2 + a + 1) = a^3 + a^2 + a$

$a^{12} = a(a^3 + a^2 + a) = a^4 + a^3 + a^2 = a^3 + a^2 + a + 1$

$a^{13} = a^4 + a^3 + a^2 + a = a^3 + a^2 + a + a + 1 = a^3 + a^2 + 1$

$a^{14} = a^4 + a^3 + a = a^3 + a + a + 1 = a^3 + 1$

$a^{15} = a^4 + a = a + a + 1 = 1$

Primitive Element

The element a in the previous table is called a primitive element of the field $GF(2^m)$. In general, any element of $GF(2^m)$ whose powers generate all the non-zero elements of $GF(2^m)$ is said to be primitive. For example, the powers of a^4 in $GF(2^4)$ given in Table A.2. are:

Table A.3. Powers of a^4 to generate non-zero elements of $GF(2^m)$

$(a^4)^0 = 1$, $(a^4)^1 = a^4$, $(a^4)^2 = a^8$,

$(a^4)^3 = a^{12}$, $(a^4)^4 = a^{16} = a$, $(a^4)^5 = a^{20} = a^5$,

$(a^4)^6 = a^{24} = a^9$, $(a^4)^7 = a^{28} = a^{13}$, $(a^4)^8 = a^{32} = a^2$,

$(a^4)^9 = a^{36} = a^6$, $(a^4)^{10} = a^{40} = a^{10}$, $(a^4)^{11} = a^{44} = a^{14}$,

$(a^4)^{12} = a^{48} = a^3$, $(a^4)^{13} = a^{52} = a^7$, $(a^4)^{14} = a^{56} = a^{11}$,

We see that the powers of a^4 generate all the fifteen non-zero elements of $GF(2^4)$. Thus, a^4 is a primitive element of $GF(2^4)$. If we attempt to see whether a^3 is a primitive element, we will find that the powers of a^3 do not generate all the elements of $GF(2^4)$. Therefore, a^3 is not a primitive element of $GF(2^4)$.

Primitive Polynomial

A polynomial p(x) of degree m that gives a complete table with 2^m distinct symbols containing 0 and 1 is called primitive.[1] Alternately, an irreducible polynomial p(x) of degree m is called primitive if p(B) = 0, where B is a primitive element of $GF(2^m)$. Thus, for m = 4,

$(x^4 + x + 1)$ is a primitive polynomial, since a^4 is a primitive element of $GF(2^4)$ and $p(a^4) = 0$.

For each positive integer m, there exists at least one primitive polynomial of degree m. It is not easy to recognize a primitive polynomial. Detailed tables of primitive polynomials are readily available and given in References 4 and 5. A few primitive polynomials are listed in Table A.4.

To Add, Multiply and Divide the Field Elements

To multiply two field elements, we simply add exponents and use the fact that $a^{15} = 1$ (or $a^{2^m-1} = 1$). To add two symbols, we use the other form in Table A.2. Example:

$$a^5 + a^7 = (a^2 + a) + (a^3 + a + 1) = a^3 + a^2 + 1 = a^{13}$$

$$1 + a^5 + a^{10} = 1 + a^2 + a + a^2 + a + 1 = 0$$

$$a^5 \cdot a^7 = a^{12}$$

$$a^{12} \cdot a^7 = a^{19} = a^4$$

Alternately,

$$a^{12} \quad a^5 = a^7$$

$$a^4 \quad a^{12} = a^{19} \quad a^{12} = 7$$

Since 1 = -1, subtraction is same as addition. The following example illustrates how to do the various Galois Field computations.

Suppose we want to solve the equation,

$$f(x) = x^2 + xa^7 + a = 0$$

The ordinary method won't work because it requires dividing by 2, and in this field 2 = 0. If f(x) = 0 has any solutions in $GF(2^4)$,

Table A.4. Primitive polynomials.

3	$1 + x + x^3$
4	$1 + x + x^4$
5	$1 + x^2 + x^5$
6	$1 + x + x^6$
7	$1 + x^3 + x^7$
8	$1 + x^2 + x^3 + x^4 + x^8$
9	$1 + x^4 + x^9$
10	$1 + x^3 + x^{10}$
11	$1 + x^2 + x^{11}$
12	$1 + x + x^4 + x^6 + x^{12}$
13	$1 + x + x^3 + x^4 + x^{13}$
14	$1 + x + x^6 + x^{10} + x^{14}$
15	$1 + x + x^{15}$
16	$1 + x + x^3 + x^{12} + x^{16}$
17	$1 + x^3 + x^{17}$
18	$1 + x^7 + x^{18}$
19	$1 + x + x^2 + x^5 + x^{19}$
20	$1 + x\, x^3 + x^{20}$
21	$1 + x^2 + x^{21}$
22	$1 + x + x^{22}$
23	$1 + x^5 + x^{23}$
24	$1 + x + x^2 + x^7 + x^{24}$

the solution can be found simply by substituting all the symbols of Table A.2 for x. The only symbols that satisfy $f(x) = 0$ are a^6 and a^{10}, since

$$f(a^6) = (a^6)^2 + a^7 \cdot a^6 + a = a^{12} + a^{13} + a = 0$$

$$f(a^{10}) = (a^{10}) + a^7 \cdot a^{10} + a = a^5 + a^2 + a = 0$$

Thus $f(x) = (x + a^6)(x + a^{10})$, where a^6 and a^{10} are roots of $f(x)$.

The above computations are very useful in the decoding of BCH codes, and they can be programmed quite easily on a general purpose computer.

Minimum Polynomial

Let $f(x) = f_k x^k + f_{k-1} x^{k-1} + \ldots + f_1 x + f_0$ where $f_i = 0$ or 1.

$$f(x) = (f_k x^k + f_{k-1} x^{k-1} + \ldots f_1 x + f_0)^2$$

$$= (f_k x^k)^2 + 2 (f_k x^k)(f_{k-1} x^{k-1} + \ldots + f_1 x + f_0)$$

$$+ (f_{k-1} x^{k-1} + \ldots + f_1 x + f_0)^2$$

Since $1 + 1 = 0$ and $1.1 = 1$ in mod 2 arithmetic,

$$f^2(x) = f_k x^{2k} + (f_{k-1} x^{k-1} + \ldots f_1 x + f_0)^2$$

$$f^2(x) = f_k x^{2k} + f_{k-1} x^{2(k-1)} + f_1 x^2 + f_0$$

$$f^2(x) = f(x^2)$$

Example:

If $f(x) = x^4 + x + 1$

$$f^2(x) = (x^4 + x + 1)^2 = x^8 + x^2 + 1 + 2x^5 + 2x + 2x^4$$

i.e. $f^2(x) = x^8 + x^2 + 1 = f(x^2)$.

Thus, for any positive integer

$$(f(x))^{2^l} = f(x)^{2^l}$$

Let B be an arbitrary element of the Galois Field $GF(2^m)$. The polynomial M(x) of the smallest degree with binary coefficients such that M(B) = 0 is called the minimum polynomial of B. The minimum polynomial of B is irreducible. Now, since M(B) = 0 and $(M(x))^{2^l} = M(x^{2^l})$, it follows that

$$(M(B)^{2^l} = M(B^{2^l}) = 0.$$

This is to say that B^{2^l} is also a root of M(x). It follows then, that

$$B,\ B^2,\ B^{2^2},\ \ldots.,\ B^{2^l},\ \ldots.$$

are all roots of M(x). Since M(x) has a finite degree, it must have a finite number of roots. Thus, there must be a repetition in the above sequence. Let e be the degree of M(x). It can be shown that $B,\ B^2,\ B^{2^2},\ \ldots.\ B^{2^{e-1}}$ are all distinct roots of M(x). These elements will repeat in sequence after $B^{2^{e-1}}$.

To Find Minimum Polynomial

The method of finding the minimum polynomial of a given element B in $GF(2^m)$ can best be illustrated by an example. Let m = 4, B = 2. We find the powers of B until the sequence starts repeating.

$$B = a \quad m\ B^2 = a^2,\ B^{2^2} = a^4,\ B^{2^3} = a^8,\ B^{2^4} = a^{16} = a.$$

Therefore, the minimum polynomial of a, $M_1(x)$ is

$$M_1(x) = (x + a)\ (x + a^2)\ (x + a^4)\ (x + a^8)$$

$$= x^4 + x^3 (a^8 + a^4 + a^2 + a) + x^2 (a^{12} + a^{10} + a^6 + a^9 + a^5 + a^3)$$
$$+ x(a^{14} + a^{13} + a^{11} + a^7) + 1$$

Substituting the values of a^i from Table A.2 (i = 1, 2,.... 14).

we get

$$M_1(x) = x^4 + x + 1$$

The minimum polynomial of a is $(x^4 + x + 1)$.

The treatment of Galois Field algebra and arithmetic done so far is enough to appreciate the BCH coding theory. As can be seen, detailed or complicated mathematical expressions and proofs have been avoided as far as possible. For further details, the texts of Peterson, W. W.[3] and Berlekamp, E. R.[2] are suggested.

APPENDIX B

PARAMETERS OF BCH CYCLIC CODES

The parameters of all binary BCH codes of length 2^m-1, with $3 \leq m \leq 9$ are given in Table B.1.*

Generator Polynomial Table**

This Table presents a table of the polynomials which are the generators of all possible nontrivial BCH codes. The polynomials are presented in octal form.

Example: The second entry in the table lists the generator g(x) as 23, which in binary form is 0 1 0 0 1 1 and therefore, the generator polynomial for a BCH code of length 15 bits, of which 11 are information bits, is $x^4 + x + 1$.

The table also shows k, the number of information bits, n, the number of bits in a codeword, and t, the maximum number of random errors which can be corrected.

* This Table is taken from Chapter 6, Page 114, of the text by Shu Lin, Reference 1.

** John P. Stenbit, IEEE Transaction on Information Theory, Vol. IT-10, October, 1964.

Table B.1. Parameters of BCH Codes.

n	k	t
7	4	1
15	11	1
	7	2
	5	3
31	26	1
	21	2
	16	3
	11	5
	6	7
63	57	1
	51	2
	45	3
	39	4
	36	5
	30	6
	24	7
	18	10
	16	11
	10	13
	7	15
127	120	1
	113	2
	106	3
	99	4
	92	5
	85	6
	78	7
	71	9
	64	10
	57	11
	50	13
	43	14
	36	15
	29	21
	22	23
	15	27
	8	31
255	247	1
	239	2
	231	3
	223	4
	215	5
	207	6
	199	7
	191	8
	187	9
	179	10
	171	11
	163	12
	155	13
	147	14
	139	15
	131	18
	123	19
	115	21
	107	22
	99	23
	91	25
	87	26
	79	27
	71	29
	63	30
	55	31
	47	42
	45	43
	37	45
	29	47
	21	55
	13	59
	9	63
511	502	1
	493	2
	484	3
	475	4
	466	5
	457	6
	448	7
	439	8
	430	9
	421	10
	412	11
	403	12
	394	13
	385	14
511	376	15
	367	16
	358	18
	349	19
	340	20
	331	21
	322	22
	313	23
	304	25
	295	26
	286	27
	277	28
	268	29
	259	30
	250	31
	241	36
	238	37
	229	38
	220	39
	211	41
	202	42
	193	43
	184	45
	175	46
	166	47
	157	51
	148	53
	139	54
	130	55
	121	58
	112	59
	103	61
	94	62
	85	63
	76	85
	67	87
	58	91
	49	93
	40	95
	31	109
	28	111
	19	119
	10	121

Table B.2. Generator Polynomials.

7	4	1	13
15	11	1	23
	7	2	721
	5	3	2467
31	26	1	45
	21	2	3551
	16	3	107657
	11	5	5423325
	6	7	313365047
63	57	1	103
	51	2	12471
	45	3	1701317
	39	4	166623567
	36	5	1033500423
	30	6	157464165547
	24	7	17323260404441
	18	10	1363026512351725
	16	11	6331141367235453
	10	13	47262230552725015 5
	7	15	52310455435032717 37
127	120	1	211
	113	2	41567
	106	3	11554743
	99	4	3447023271
	92	5	624730022327
	85	6	130704476322273
	78	7	26230002166130115
	71	9	625501071325312775 3
	64	10	1206534025570773100045
	57	11	33526552525057050535177 21
	50	13	54446512523314012421501421
	43	14	17721772213651227521220517434 3
	36	15	31460746665220750447645747217 35
	29	31	40311446136767060366753014117615 5
	22	23	12337607040472252243544562663764704 3
	15	27	22057042445604554770523013762217604353
	8	31	70472640527510306514762242715677331302 17

APPENDIX C

ENCODER AND DECODER CIRCUITS

The Encoder Circuit

With reference to Figure C-1, the output ABCD of Inverter UIe is 1 for the first seven counts of the divide by fifteen counter, i.e., the output is 1 for ABCD = 0 0 0 0 to 0 1 1 0. For the next eight counts, i.e., for ABCD = 0 1 1 1 to 1 1 1 0, the inverter UIe has a zero output.

When the first 7 bits are fed in the encoder, Nand gate U2c is on and is off for the next eight pulse counts. The Nand gate U2d, similarly, is on during the last eight counts, so that the output appears through pin 4 of the Exclusive-OR gate u1b. The trigger for the pulse is obtained through a manually operated, practically bounce-free, micro-switch. The pulse is generated by the Astable multivibrator, U1.

The Decoder Circuit

The decoder circuit appears in Figures C-2, C-3 and C-4. The counts are controlled with the divide-by-32 counter and a combination logic circuit. For the first 15 counts, i.e., when the counter reading is from 0 0 0 0 0 to 0 1 1 1 1, Nand gate U3a is on and is off for the next 16 counts. The power sums S_1 and S_3 are formed after the fifteen bits of the received word are fed into the S_1 and S_3 registers.

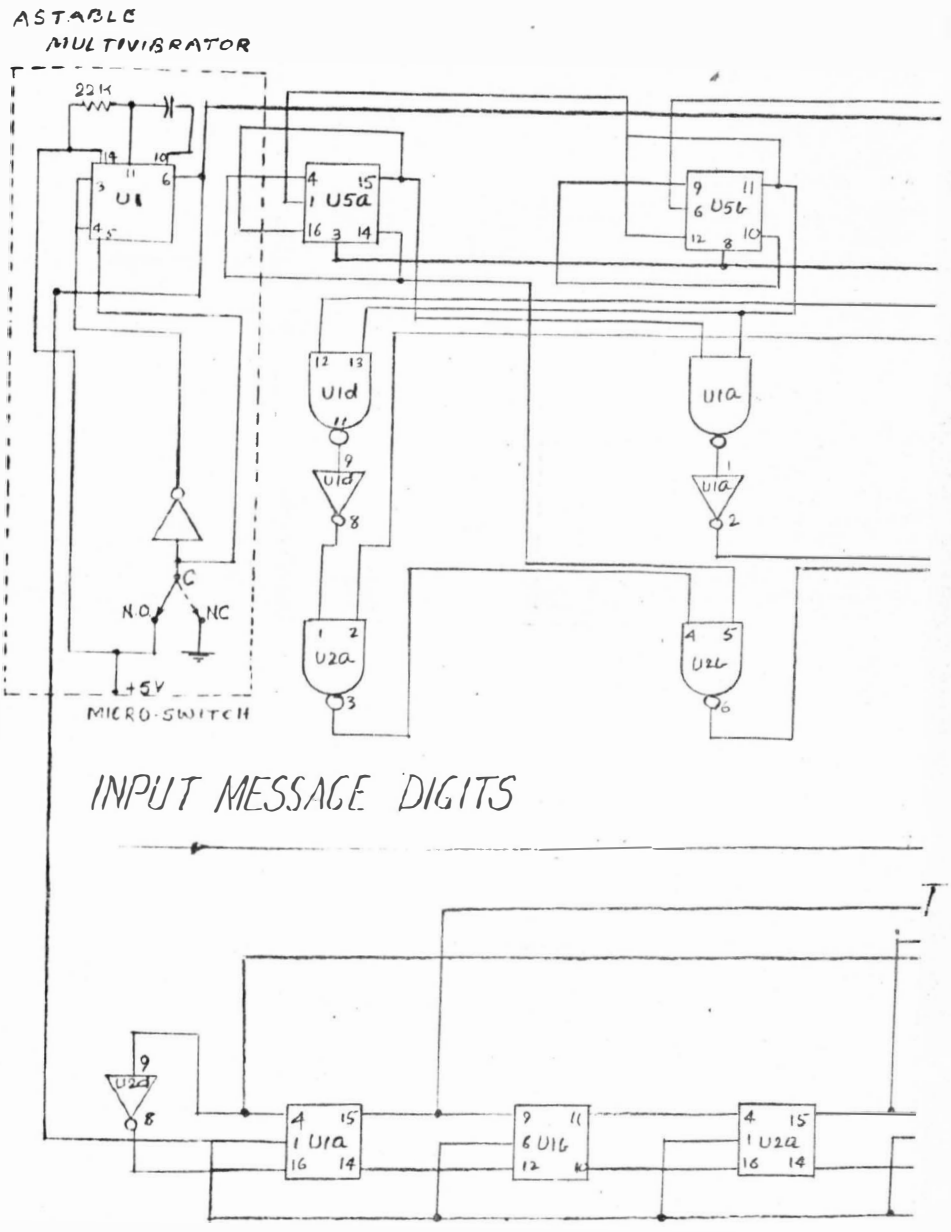


FIGURE C1 K-STAGE ENCODER FOR A

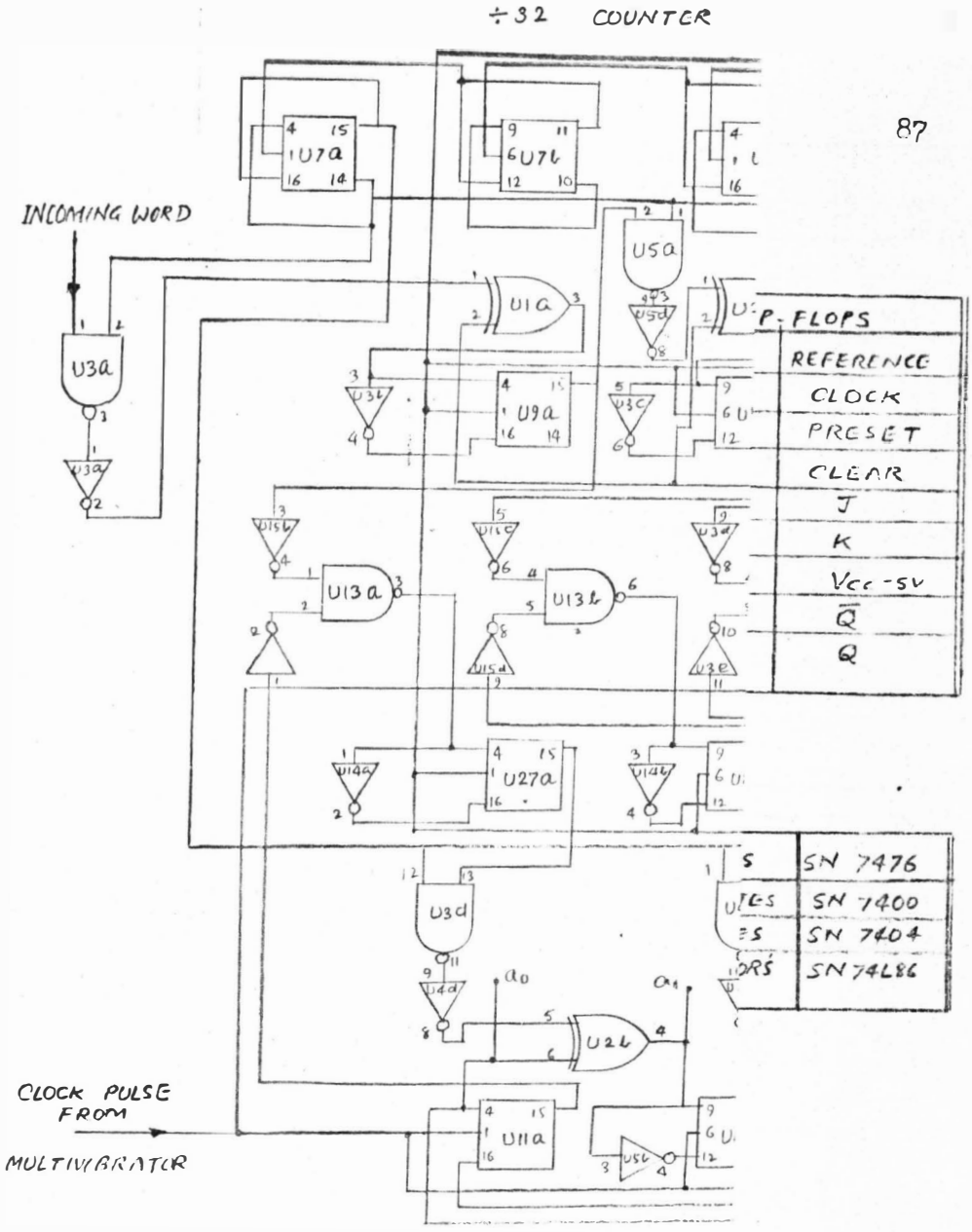


FIGURE C2 S_1 AND α-MULTIPL

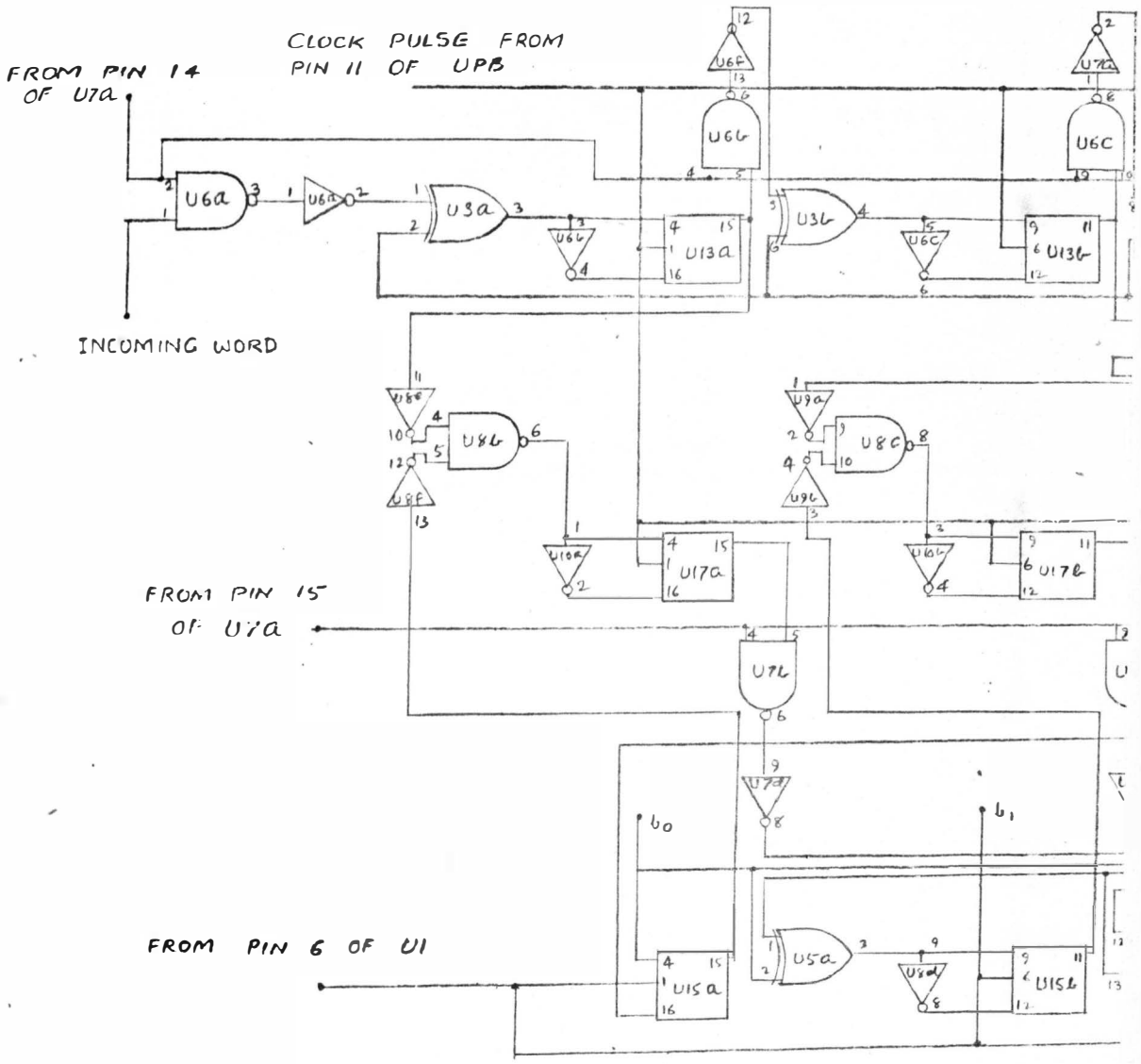


FIGURE C3 S_3 AND α^3-MULTIPLIER C.

On the sixteenth pulse, Nand gates U3d, U4a, U4b, and U4c turn on and S_1 and S_3 are transferred to the S_1 and S_3 storage registers. During the next pulse, the S_1 and S_3 storage registers remain unchanged, whereas the a-multiplier and a^3-multiplier registers contain the product S_1a and S_3a^3 respectively. This is made possible by giving a clock frequency to the a and a^3 multiplier circuits, double of that to the S_1 and S_3 registers. The product S_1a and S_3a^3 is fed to the cyclic error correction unit. The output of this unit and the outgoing bit from the buffer combine together to change a bit if it is in error.

On the next clock pulse, S_1a and S_3a^3 are stored in the S_1 and S_3 storage registers and these contents are multiplied again during the next pulse to a and a^3 multipliers, by a and a^3 respectively. This process is continued until all the 15 bits have been sent through the cyclic error correction unit and corrected if errors are present. It is to be noted here that the use of master-slave type j-k flip-flops in this device eliminates erroneous operations even if the clock preset and clear pulses are distorted.

Power Supply for Lamps

A separate power supply was designed for the lamps. The circuit appears in Figure C.5. Four Westinghouse 1N1200A diodes with a current capacity of 5 amps were used, together with a transistor (n-p-n 2N708) with a beta of over 50. If the logic 1 is available at the base of the transistor, the lamp goes on, drawing only negligible current

from the supply to the IC modules. If a logic zero is fed into the base of the transistor, the lamp turns off. The total number of gates required for the entire circuit is as follows.

Gates	Quantity
Shift Registers	58
Nand Gates	56
Inverters	84
Exclusive-OR Gates	36

The total cost of the entire project is estimated at about $200. The layout of components is shown in Figure C.6.

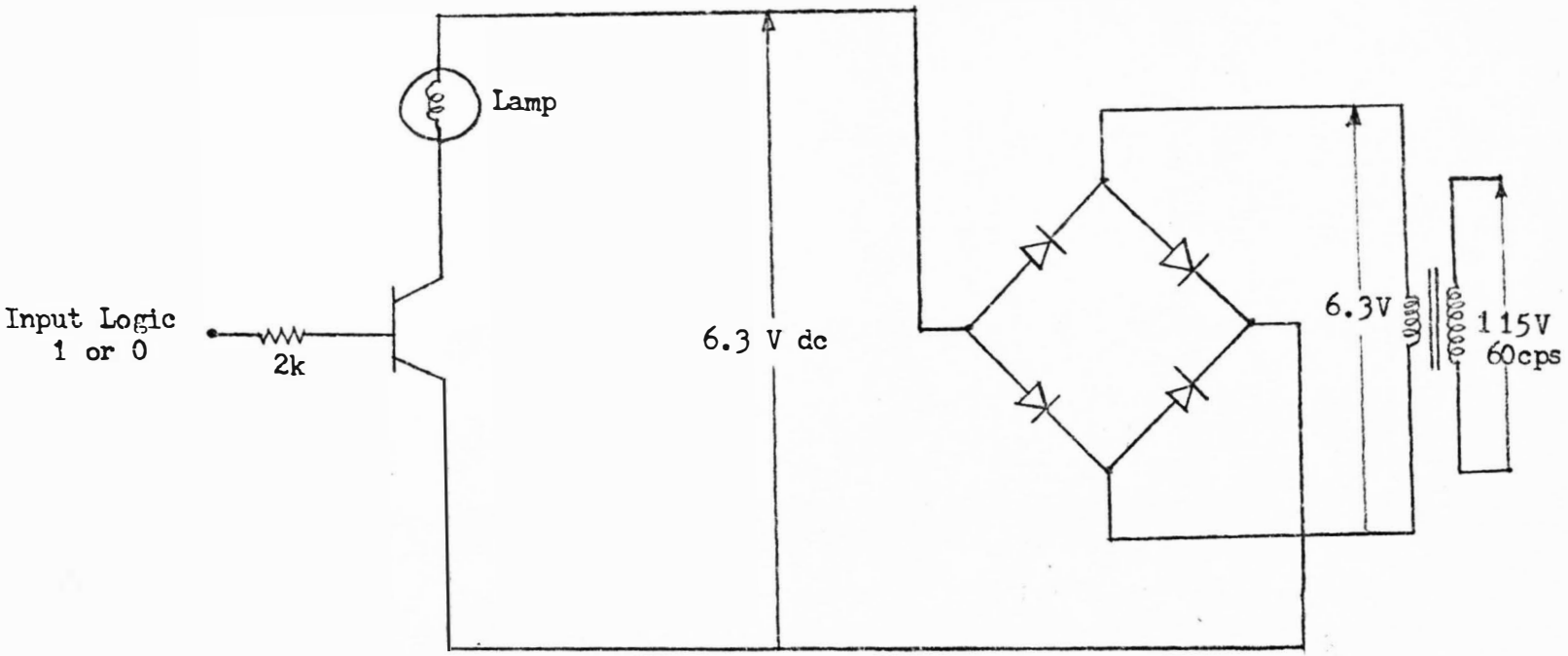


Figure C5 DC power supply for lamps.

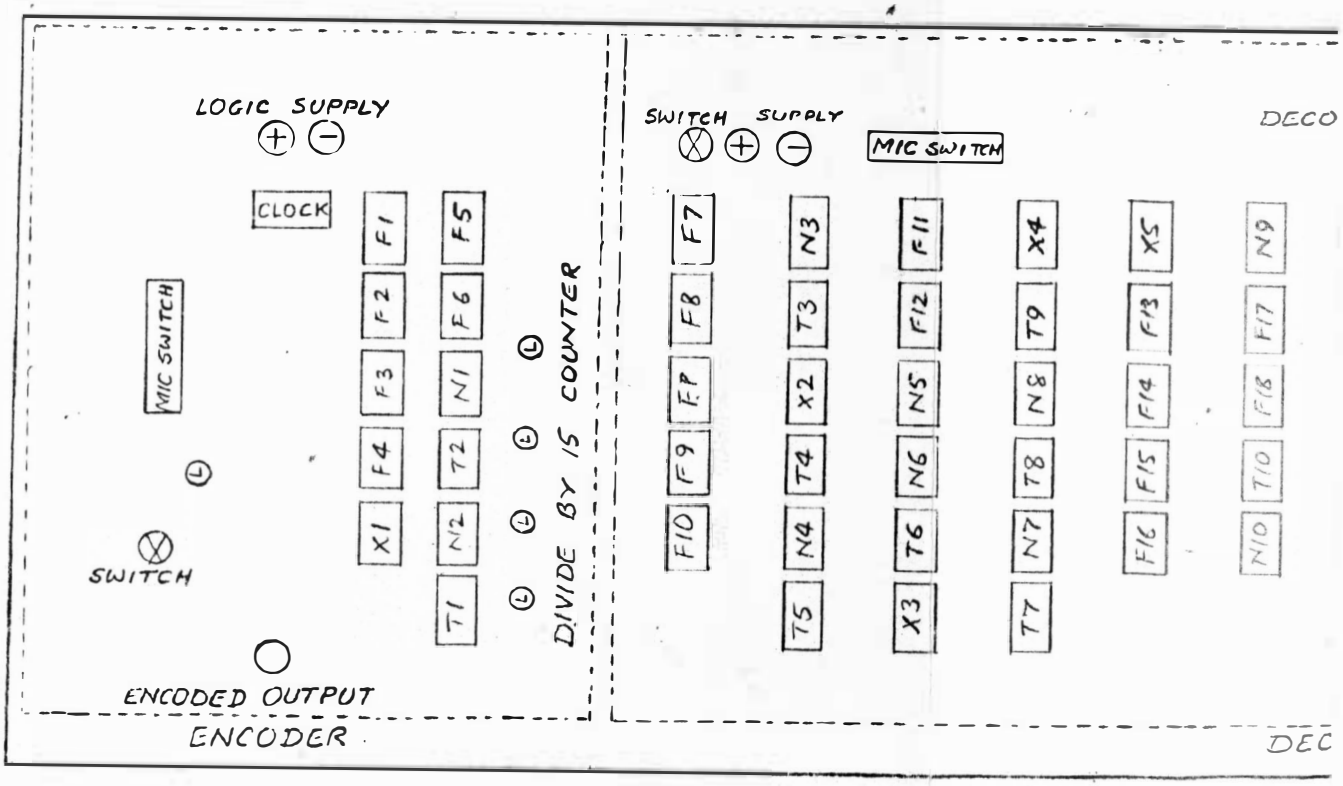
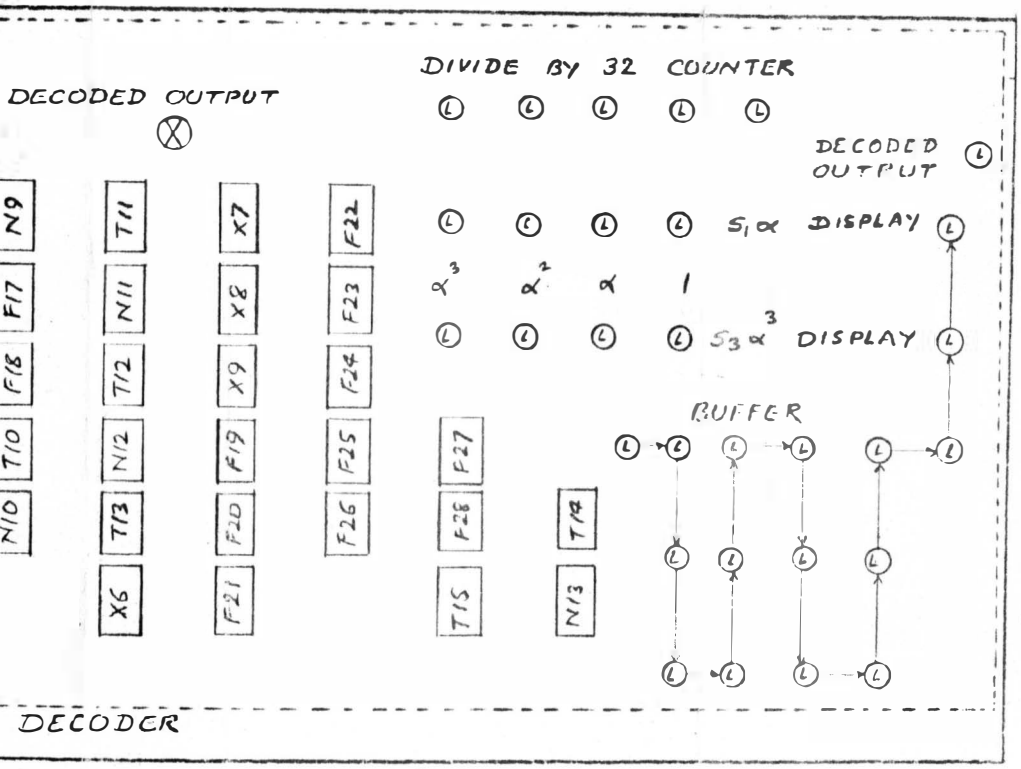


FIGURE C6

COMPONENT LAYOUT OF THE CIRCUI

FIGURE NOT TO SCALE



RCUIT

SYMBOL	DESCRIPTION	NO.
X1, X2	EXCLUSIVE-OR GATES	SN 7486
FF1, FF2	J-K MASTER SLAVE FF'S	SN 7476
N1, N2	2-INPUT NAND GATES	SN 7400
T1, T2	HEX INVERTERS	SN 7404